Introduction to Real-Time Process Scheduling

Q: Many theories and algorithms in real-time process scheduling seem to have simplified assumptions without direct solutions to engineers’ problems. Why should we know them?

A:
◆ Provide insight in choosing a good system design and scheduling algorithm.
◆ Avoid poor or erroneous choices.
Introduction to Real-Time Process Scheduling

- Checklist
  ⊕ What do we really know about the rate monotonic (RM) and the earliest deadline first (EDF) scheduling?
  ⊕ What is known about uniprocessor real-time scheduling problems?
  ⊕ What is known about multiprocessor real-time scheduling problems?
  ⊕ What is known about energy-efficient real-time scheduling problems?
  ⊕ What task-set characteristics cause NP-hard?
  ⊗ What is the impact of overloads on the scheduling results?
  ⊗ What do we really know about theories for off-line schedulability such as the rate monotonic analysis?

---

<table>
<thead>
<tr>
<th>Time</th>
<th>Job Shop Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Independent Process Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Liu &amp; Layland, 1973, etc.)</td>
</tr>
<tr>
<td></td>
<td>Process Scheduling with Non-Preemptable Resources</td>
</tr>
<tr>
<td></td>
<td>Process Scheduling with End-to-End Delays</td>
</tr>
<tr>
<td></td>
<td>(Stankovic, Gerber, Lin, etc, since ?)</td>
</tr>
<tr>
<td></td>
<td>Process Scheduling with Multiple Resources</td>
</tr>
<tr>
<td></td>
<td>(??)</td>
</tr>
<tr>
<td></td>
<td>Energy-Efficient Process Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Morse, Hakan Chen, Kuo, etc)</td>
</tr>
<tr>
<td></td>
<td>Non-preemptable Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Baruah, 1990, etc.)</td>
</tr>
<tr>
<td></td>
<td>Process Scheduling with Probabilistic Guarantee</td>
</tr>
<tr>
<td></td>
<td>(Liu, Lehoczky, etc, since 1995)</td>
</tr>
<tr>
<td></td>
<td>Process Scheduling with Realistic Task Characteristics</td>
</tr>
<tr>
<td></td>
<td>(Liu, Mok, etc, since 1996)</td>
</tr>
<tr>
<td></td>
<td>Rate-Based Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Butazzo, Liu, Brauah, Kuo, etc, since 1995)</td>
</tr>
<tr>
<td></td>
<td>Multiprocessor Process Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Baruah, Anderson, etc)</td>
</tr>
<tr>
<td></td>
<td>Multiprocessor Process Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Dhall, 1972, etc)</td>
</tr>
<tr>
<td></td>
<td>Sporadic Process Scheduling</td>
</tr>
<tr>
<td></td>
<td>(Sprunt, 1989, etc)</td>
</tr>
</tbody>
</table>

---
Introduction to Real-Time Process Scheduling

Uniprocessor Process Scheduling

- Rate Monotonic Scheduling
- Earliest Deadline First Scheduling
- Priority Ceiling Protocol
- Important Theories


Process Model

- Periodic process
  - each periodic process arrives at a regular frequency - a special case of demand.
    - r: ready time, d: relative deadline, p: period, c: maximum computation time.
  - For example, maintaining a display

- Sporadic process
  - An aperiodic process with bounded inter-arrival time p.
  - For example, turning on a light

- Other requirements and issues:
  - process synchronization including precedence and critical sections, process value, etc.
Performance Metrics

- Metrics for hard real-time processes:
  - Schedulability, etc.
- Metrics for soft real-time processes:
  - Miss ratio
  - Accumulated value
  - Response time, etc.
- Other metrics:
  - Optimality, overload handling, mode-change handling, stability, jitter, etc.
  - Combinations of metrics.

Basic definitions:

- **Preemptive scheduling**: allows process preemptions. (vs non-preemptive scheduling)
- **Online scheduling**: allocates resources for processes depending on the current workload. (vs offline scheduling)
- **Static scheduling**: operates on a fixed set of processes and produces a single schedule that is fixed at all time. (vs dynamic scheduling)
- **Firm real-time process**: will be killed after it misses its deadline. (vs hard and soft real-time)
- **Fixed-priority scheduling**: in which the priority of each process is fixed for any instantiation. (vs dynamic-priority scheduling)
Rate Monotonic Scheduling Algorithm

- Assumptions:
  - all periodic fixed-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Rate Monotonic (RM) Scheduling Algorithm
  - RM priority assignment: priority ~ 1/period.
  - preemptive priority-driven scheduling.
- Example: T1 (p1=4, c1=2) and T2 (p2=5, c2=1)

Critical Instant

- An instant at which a request of the process have the largest completion/response time.
- An instance at which the process is requested simultaneously with requests of all higher priority processes

Usages

- Worst-case analysis
- Fully utilization of the processor power
- Example: T1 (p1=4, c1=2) and T2 (p2=5, c2=1 → 2)
Rate Monotonic Scheduling Algorithm

- Schedulability Test:
  - A sufficient but not necessary condition
  - Achievable utilization factor $\alpha$ of a scheduling policy $P$ -> any process set with total utilization factor $\sum \frac{c_i}{p_i}$ no more than $\alpha$ is schedulable.
  - Given n processes, $\alpha = n\left(2^{1/n} - 1\right)$
- Stability:
  - Let processes be sorted in RM order. The ith process is schedulable if $\sum_{j=1}^{i} \frac{c_j}{p_j} \leq \left(2^{1/i} - 1\right)$
- An optimal fixed priority scheduling algorithm

Rate Monotonic Scheduling Algorithm

- Rate Monotonic Analysis (RMA) $^2$
  - Basic Idea:
    Before time $t$ after the critical instance of process $\tau_i$, a high priority process $\tau_j$ may request $\left[ \frac{c_j t}{p_j} \right]$ amount of computation time.
  - Formula:
    $$W_i(t) = \sum_{j=1}^{i} c_j \left[ \frac{t}{p_j} \right] \leq t \leq d_i \quad \text{for some } t \text{ in } \{ kp_j | j = 1, ..., i; k = 1, ..., \left[ \frac{p_j}{p_i} \right] \}$$
  - A sufficient and necessary condition and many extensions...

$^2$ Sha, "An Introduction to Rate Monotonic Analysis," tutorial notes, SEI, CMU, 1992
Rate Monotonic Scheduling Algorithm

- A RMA Example:
  - T1(20,100), T2(30,150), T3(80, 210), T4(100,400)
  - T1
    - c1 <= 100
  - T2
    - c1 + c2 <= 100 or
    - 2c1 + c2 <= 150
  - T3
    - c1 + c2 + c3 <= 100 or
    - 2c1 + c2 + c3 <= 150 or
    - 2c1 + 2c2 + c3 <= 200 or
    - 3c1 + 2c2 + c3 <= 210
  - T4
    - c1 + c2 + c3 + c4 <= 100 or
    - 2c1 + c2 + c3 + c4 <= 150 or
    - ....

RM was chosen by
- Space Station Freedom Project
- FAA Advanced Automation System (AAS)

RM influenced
- the specs of IEEE Futurebus+

RMA is widely used for off-line analysis of time-critical systems.
Earliest Deadline First Scheduling Algorithm

- Assumptions (similar to RM):
  - all periodic dynamic-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources

- Earliest Deadline First (EDF) Scheduling Algorithm:
  - EDF priority assignment: priority ~ absolute deadline.
    i.e., arrival time $t + \text{relative deadline } d$. 
  - preemptive priority-driven scheduling

- Example: $T_1(c_1=1, p_1=2)$, $T_2(c_2=2, p_2=7)$

```
+---+---+---+---+---+---+---+---+
| T1 | T2 | T1 | T2 | T1 | T1 | T2 |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  |
```

Earliest Deadline First Scheduling Algorithm

- Schedulability Test:
  - A sufficient and necessary condition
  - Any process set is schedulable by EDF iff
    \[
    \sum_{j=1}^{i} \frac{C_j}{P_j} \leq 1
    \]

- EDF is optimal for any independent process scheduling algorithms

- However, its implementation has considerable overheads on OS’s with a fixed-priority scheduler and is bad for (transiently) overloaded systems.
Priority Ceiling Protocol

Assumptions (as the same as RM for the first two):
- all periodic fixed-priority processes
- relative deadline = period
- Non-preemptable resources guarded by semaphores

Basic Ideas and Mechanisms:
- Bound the priority inversions by early blocking of processes that could cause them, and
- Minimize a priority inversion’s length by allowing a temporary rise in the blocking process’s priority.

Contribution of the Priority Ceiling Protocol
- Efficiently find a suboptimal solution with a clever allocation policy, guaranteeing at the same time a minimum level of performance.

Pre-requirements: nested critical sections!

Priority Ceiling Protocol (PCP):
- Define a semaphore’s priority ceiling as the priority of the highest priority process that may lock the semaphore.
- Lock request for a semaphore is granted only if the requesting process’s priority is higher than the ceiling of all semaphores concurrently locked by other processes.
- In case of blocking, the task holding the lock inherits the requesting process’s priority until it unlocks the corresponding semaphore. (Def: priority inheritance)

Priority Ceiling Protocol

- A PCP Example: deadlock avoidance

\[ \begin{align*}
\tau_1 &: \text{attempt to lock } S_2 \\
\tau_2 &: \text{priority inheritance}
\end{align*} \]

- A PCP Example: avoid chain blocking

\[ \begin{align*}
\tau_0 &: \text{attempt to lock } S_2 \\
\tau_1 &: \text{priority inheritance}
\end{align*} \]
Priority Ceiling Protocol

- **A PCP Example:** one priority inversion

\[ \tau_0 \]
\[ \tau_1 \]
\[ \tau_2 \]

**Important Properties:**
- A process is blocked at most once before it enters its critical section.
- PCP prevents deadlocks.

**Schedulability Test of \( \tau_i \)**
- worst case blocking time \( B_i \) - an approximation!
  - \( S_j = \{ S \mid \text{semaphore } S \text{ is accessed by } \tau_j \} \)
  - \( BS_i = \{ \tau_j \mid j > i \text{ and } Max(\text{ceiling}(s)) \geq \text{priority}(\tau_i) \} \)
  - \( B_i = Max(\text{in } BS_i) \{ \text{critical section} \} \)
- Let processes be sorted in the RM priority order

\[
\sum_{j=1}^{i-1} \left( \frac{C_j}{P_j} \right) + \frac{C_i + B_i}{P_i} \leq i \left( 2^{1/i} - 1 \right)
\]
Priority Ceiling Protocol

Variations of PCP:
- Stack Resource Policy - not permitted to start unless resources are all available.
  - multi-units per resource
  - dynamic and fixed priority assignments
- Dynamic Priority Ceiling Protocol
  - extend PCP into an EDF scheduler.

Introduction to Real-Time Process Scheduling

Multiprocessor Process Scheduling

- Important Theories
- Basic Approaches

Multiprocessor Process Scheduling

- Checklist
  - Understand the boundary between polynomial and NP-hard problems to provide insights into developing useful heuristics.
  - Understand the fundamental limitations of on-line algorithms to create robust system and avoid misconceptions and serious anomalies.
  - Know the basic approaches in solving multiprocessing scheduling

- Remark: It is the area which we have very limited knowledge because of its complexity and our minimal experiences with multiprocessor systems.

Nonpreemptive Multiprocessor Scheduling

- Important Theorems\(^1\):
  - Conditions:
    - Single deadline, identical processors, ready at time 0
  - Theorems: ("_"-marked items causes NP-completeness!)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Resources</th>
<th>Ordering</th>
<th>Computation Time</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>Arbitrary</td>
<td>Unit</td>
<td>Polynomial(^2)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Independent</td>
<td>Arbitrary</td>
<td>NP-Complete(^3)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Arbitrary</td>
<td>1 or 2 units</td>
<td>NP-Complete(^3)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Forest</td>
<td>Unit</td>
<td>NP-Complete(^3)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Independent</td>
<td>Unit</td>
<td>NP-Complete(^3)</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>Forest</td>
<td>Unit</td>
<td>Polynomial(^4)</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>Arbitrary</td>
<td>Unit</td>
<td>NP-Complete(^5)</td>
</tr>
</tbody>
</table>

Preemptive Multiprocessor Scheduling

- Theorem of McNaughton in 1959.
  - Goal: Compare preemption and non-preemption.
  - Conditions:
    - Processors.
  - Theorem 0: Given the metric to minimize the weighted sum of completion times, i.e., \( \text{Sum}(w_iC_i) \), there exists a schedule with no preemption for which the performance is as good as for any schedule with a finite number of preemptions.
  - Note: It is NP-hard to find an optimal schedule! If the metric is to minimize the sum of completion times, the shortest-processing-time-first greedy approach is optimal.


Preemptive Multiprocessor Scheduling

- Theorem of Lawler in 1983.
  - Goal: Show that heuristics are needed for real-time multiprocessor scheduling.
  - Conditions:
    - Processes.
  - Theorem 0: The multiprocessing problem of scheduling \( P \) processors with process preemption allowed and with minimization of the number of late processes is NP-hard.

Preemptive Multiprocessor Scheduling

- Theorems of Mok in 1983
  - Goal: Understand the limitations of EDF.
  - Conditions:
    - different ready times.
  - Theorem 0: Earliest-deadline-first scheduling is not optimal in the multiprocessor case.
  - Example, T1(c=1,d=1), T2(c=1,d=2), T3(c=3,d=3.5), two processors.
  - Theorem 1: For two or more processors, no deadline scheduling algorithm can be optimal without complete a priori knowledge of deadlines, computation times, and process start times.

Multiprocessor Anomalies

  - Goal: Notice anomaly and provide better design.
  - Conditions:
    - A set of processes is optimally scheduled on a multiprocessor with some priority order, fixed execution times, precedence constraints, and a fixed number of processors.
  - Theorem 0: For the stated problem, changing the priority list, increasing the number of processors, reducing execution times, or weakening the precedence constraints can increase the schedule length.
Multiprocessor Anomalies

- An Example

```
P1   P2
[ ]  [ ]
[ ]  [ ]
```

```
P1   P2
[ ]  [ ]
[ ]  [ ]
```

Motivation:
- The multiprocessor scheduling problem is NP-hard under any but the most simplifying assumptions.
- The uniprocessor scheduling problem is usually tractable.

Common Approach - 2 Steps
- Assign processes to processors
- Run a uniprocessor scheduling algorithm on each processor.

Metrics:
- Minimize the number of processors, fault tolerance, etc.
Multiprocessor Scheduling - Contemporary Approach

- However, the process assignment problem is again NP-hard in most cases.
- Heuristics:
  - Utilization balancing - balance workload of processors.\(^1\)
  - Next-fit algorithm - used with RM. \(^2\)
  - Bin-packing algorithm - set with a threshold and used with EDF \(^3\), etc.
- Other considerations:
  - precedence constraints, dynamic overload handling, etc.

---


---

Multiprocessor Scheduling

- Current Research
  - Classification: Migration(/Partition) & Static or Dynamic Priorities
  - Some Recent Results:
    - Utilization Bound = 42% by a bin-packing partitioning approach (JRTS, 1999)
    - Utilization Bound = 37.482% by RM-US – processes with a utilization > bound is given the highest priority; otherwise RM is adopted.
    - Utilization Bound = m – [(m-1)*Umax] if Umax <= 0.5, where Umax = max Ui. Or Utilization Bound = (m+1)/2+Umax if Umax > 0.5 – M-CBS (RTAS02)
    - Utilization Bound = 75% - EZDL (to appear)
Introduction

- Challenges in Embedded Systems Designs
  - Limited Resources
  - Limited Energy Supply
  - Variety in Product Designs
  - Strong Demands in Friendly User Interface
  - Strong Mutual Influence Between Hardware and Software Designs
  - Limited Lifetime in Many Products
Introduction

- Worlds are Getting More and More Complicated!
  - Processors with Voltage-Scaling Supports
  - I/O Devices with Different Voltage Supplies and Operating Modes
  - Communication Devices with Different Operating Modes
- Where To Save Energy Consumption?
  - Hardware Designs
  - Operating Systems/System Components Designs
  - Application Systems/Programs Designs

System Design Issues – Energy-Efficiency Designs

- Operating System Designs
- Application Program Designs
- Application System Designs
Operating System Designs

- Proper Voltage-Scaling Scheduling
  - HW Architectures, Task Characteristics, etc.
  - Task Scheduling, Multi-Resource Scheduling, etc.
- Intelligent Event Management
  - Idle Time, Synchronization, Multi-Event Waiting, etc.
- Intelligent Device Management
  - Device Status Scheduling, Request Scheduling, Polling-Style Programming, etc.

The Idle Task – uC/OS-II

- The idle task is always the lowest-priority task and cannot be deleted or suspended by user tasks.
- To reduce power dissipation, you can issue a HALT-like instruction in the idle task.
  - Suspend services in OSTaskIdleHook()!!

```c
void OS_TaskIdle (void *pdata)
{
    #if OS_CRITICAL_METHOD == 3
        OS_CPU_SR cpu_sr;
    #endif

    pdata = pdata;
    for (; ; ) {
        OS_ENTER_CRITICAL();
        OSIdleCtr++;
        OS_EXIT_CRITICAL();
        OSTaskIdleHook();
    }
}
```
Application System Designs

- Application Characteristics
  - Assumptions, Optimization Goals, Architecture Choices, Topology Constraints, etc.
- Standard Constraints
  - Design Flexibility, Restrictions, Quality-of-Services, etc.
- Cross-Layer Optimization
  - Coupling Strength, Modularity, Upgradeability, etc.

Example Application System Designs – Adaptive Sensor Networks

- Power Management
  - Consider power-efficiency for node and protocol designs.
- Mobility Management
  - Detect and manage nodes with dynamic movements.
- Task Management
  - Schedule and balance workloads performed by nodes.
Application Program Designs

- Application Characteristics
  - Design Logics, Hardware Supports, Resource Utilization & Patterns, etc.
- User/Process Behaviors
  - Bottleneck Identification, Program Structures, User Access Patterns, etc.

Case Study – Energy-Profilng of a Browser

- Konqueror
  - An open-source web browser running on Linux built upon a Qt application environment
  - A full-featured web browser
    - HTML 4 compliance
    - Cascading Style Sheets (CSS1) and CSS2
    - JavaScript
    - Java Applet
    - Flash
    - SSL, etc.

- Overheads
  - The profiling overheads were no more than 7% of the profiling system (Profiling Frequency = 20,000HZ).

IEEE WORDS 2004
Motivations on Energy-Efficient Process Scheduling

- Energy-Efficiency Considerations for Battery-Powered Embedded Systems
  - Operating Duration
  - Performance
- Dimensions in Problem Formulation
  - Architecture Considerations, e.g., Homogeneous/Heterogeneous Multiprocessors
  - Process Models, e.g., Frame-Based Process Sets
  - Processor Types, e.g., Available Processor Speeds

Definitions – Voltage Scaling and Power Consumption

- A Dynamic Voltage/Speed Scaling (DVS) system is a system that can execute tasks at different speeds.
  - A higher supply voltage results in a higher frequency (or higher execution speed).
    \[ s = k \times \frac{(V_{dd} - V_t)^2}{(V_{dd})^2} \]
    - s is the corresponding speed of the supply voltage \( V_{dd} \), and \( V_t \) is the threshold voltage
  - The dynamic power consumption function \( P() \) of the execution speeds is a convex function:
    \[ P(s) = C_{ef} V_{dd}^2 s \]
    \[ P(s) = C_{ef} s^3/k^2, \text{ when } V_t = 0 \]

Example Voltage Scaling Processors:
Intel XScale, StrongARM, Transmeta, Intel Centrino
Dilemma – Performance versus Energy Consumption

- Definition: Energy-Efficient Process Scheduling
  - Given a process set with timing constraints and a set of processors with available processor speeds (and constraints), find a feasible schedule such that the energy consumption is minimized.

Task Models under Investigation

- Frame-Based Real-Time tasks
  - All the tasks are ready at time 0 and share a common deadline $D$.
  - Each task $\tau_i$ is associated with $c_i$ amount of computation requirements.
- Periodic Real-Time Tasks
  - The job of each task $\tau_i$ arrives periodically in a period $p_i$ after the first job of $\tau_i$ releases at time $a_i$.
  - Each task $\tau_i$ is associated with $c_i$ amount of computation requirements.
  - The relative deadline of $\tau_i$ is equal to $p_i$. 
Task Models under Investigation

- Aperiodic Real-Time Tasks
  - The job of each task $\tau_i$ might arrive with a minimum separation time $p_i$ after the first job of $\tau_i$ releases at time $a_i$.
  - Each task $\tau_i$ is associated with $c_i$ amount of computation requirements.
  - The relative deadline of $\tau_i$ is given as a constant $d_i$.

- Periodic Multi-Framed Real-Time Tasks
  - Periodic Real-Time Tasks
  - Each task $\tau_i$ is associated with a regular pattern of $c_i$ amount of computation requirements in successive periods.

Our Roadmap on Energy-Efficient Process Scheduling

- Energy-Efficient Task Scheduling and Synthesis
  - Task Scheduling on A Single Processor
    - Minimization on the Energy Consumption
      - Discrete Number of Processor Speeds
    - Maximization on the System Reward/Performance
      - Allow Speed Transition During Execution of Any Job
  - Task Scheduling for Multiprocessors
    - CMP Processors
      - No Speed Transition During Execution of Any Job
    - General-Purposed Processors
      - Tasks Have the Same Power Consumption Functions
      - Tasks Have Different Power Consumption Functions
Potential Directions

- Realistic Task Models
- Leakage Current
- Process Synchronization
- Multi-Core System Architectures
- I/O Peripheral Considerations
- Complicated System Architectures

Papers to Study

- http://140.112.28.119
Papers to Study