“Priority assignment in real-time systems using fixed priority scheduling”

Robert Davis

Real-Time Systems Research Group, University of York

rob.davis@york.ac.uk
What is a Real-Time System?

Real-Time System is any system which has to respond to externally generated input stimuli within a specified time:

- Functionally correct – the right computations
- Temporally correct – completed within predefined time constraints
- Time constraints typically expressed in terms of deadlines on the elapsed time between the stimuli and the corresponding response

- **Hard Real-Time**
  - Failure to meet a deadline constitutes a failure of the application (e.g. flight control system)

- **Soft Real-Time**
  - Latency in excess of the deadline leads to degraded quality of service (e.g. data acquisition, video playback)
Examples of Real-Time Systems

Robotics and Factory Automation

Instrumentation

Avionics

Telecommunications

Automotive Electronics

Space

Medical Systems
Real-Time Applications

- **Time-triggered**
  - Monitoring and data acquisition
  - Control loops
  - Typically periodic behaviours e.g. every 20ms

- **Event-triggered (interrupt-driven)**
  - Simple sensors (switch closes)
  - Engine rotation
  - Peripheral devices (e.g. comms – message received)
  - Often generate non-periodic behaviours

- Applications decomposed into
  - Multiple tasks with timing constraints
Task Timing Behaviour

- Types of task (time-triggered and event-triggered) based on pattern of arrivals
  - Periodic: generates jobs with a strict period of $T_i$ between them
  - Sporadic: minimum inter-arrival time $T_i$ between jobs
  - Aperiodic: no minimum inter-arrival time, so jobs can arrive arbitrarily close together
Task Timing Behaviour

- **Execution times**
  - Assume a bounded WCET $C_i$ for Hard Real-Time task $\tau_i$

- **Types of Deadline**
  - Implicit: Same as period / minimum inter-arrival time $D_i = T_i$
  - Constrained: $D_i \leq T_i$
  - Arbitrary: not related to $T_i$ (but needs to be $\geq C_i$)
Uniprocessor Real-Time Scheduling

Why do we need scheduling at all?
- Single processor can only execute one job at a time
- Tasks can have very different timing characteristics (C,D,T)
- Multiple tasks and each task can potentially generate an infinite sequence of jobs

Terminology
- **Scheduler**: part of a RTOS which decides at run-time which job to execute
- **Scheduling policy**: rules used by the scheduler to choose between jobs
- **Schedulability analysis**: some maths used offline to determine if jobs can always be guaranteed to meet their deadlines according to a system and task model
Scheduling policies
Fixed Priority (Pre-emptive)

Task A

Task B

Task C

8
What is this talk about?

- **Fixed Priority scheduling** in all its guises
  - Pre-emptive, non-pre-emptive, deferred pre-emption
  - Single processor, sporadic tasks

- **Priority assignment**
  - Why is it important?
  - What is an optimal priority assignment?
  - How do we find it?
  - Is Optimal Priority Assignment enough?
    - Can we optimise other things as well?
  - Unsolved priority assignment problems
Priority assignment

Why is priority assignment important

- Achieve a schedulable system when it otherwise wouldn’t be
- Provide a schedulable system avoiding hardware overprovision / maximising use of hardware resources
- Provide headroom for unforeseen interference or overruns

Example

- Controller Area Network (CAN)
- Used for in-vehicle networks
- Message IDs are the priorities
- Analysis of network schedulability resembles that of a single processor with fixed priority non-pre-emptive scheduling
When priority assignment goes bad!

- From Darren Buttle’s Keynote at ECRTS 2012

The myth of CAN bus Utilisation - “You cannot run CAN reliably at more than 30% utilisation”

1 Figures may vary but not significantly

Why?

- Message IDs i.e. priorities assigned in an ad-hoc way reflecting data and ECU supplier (legacy issues)
- ...as well as many other issues, including device driver implementation
When priority assignment goes bad!

- **Example: CAN**
  - Typical automotive config:
    - 80 messages
    - 10ms - 1s periods
    - All priority queues
  - x10,000 message sets

- **Breakdown utilisation**
  - Scale bus speed to find util. at which deadlines are missed

  80% v 30% or less

---

System model

- Single processor, fixed priority scheduling
  - Scheduler chooses the highest priority ready task to execute

- Periodic / Sporadic task model
  - Static set of $n$ tasks. Each task $\tau_i$ has a unique priority $i$
    - $C_i$ - Execution time (bound)
    - $D_i$ - Relative deadline
    - $T_i$ - Minimum inter-arrival time or period

- Variations
  - Implicit / constrained / arbitrary deadlines
  - Pre-emptive / non-pre-emptive / deferred pre-emption scheduling
  - Unique priorities or shared priority levels
Schedulability

- Schedulability tests
  - Determine if all jobs of a task (all tasks) can be guaranteed to meet their deadlines for all valid arrival patterns
  - **Sufficient** if all of the tasksets that the test deems to be schedulable are in fact schedulable
  - **Necessary** if all of the tasksets that the test deems to be unschedulable are in fact unschedulable
  - **Exact** implies both sufficient and necessary

- Worst-case response times
  - Schedulability tests often compute the worst-case response time $R_i$ for each task and compare it with the task’s deadline $D_i$ to determine schedulability
Definition: Optimal priority assignment policy

For a given system model, a priority assignment policy $P$ is referred to as \textbf{optimal} if there are no systems, compliant with the model, that are schedulable using another priority assignment policy that are not also schedulable using policy $P$.

An optimal priority assignment policy can schedule any system that can be scheduled using any other priority assignment.

May also consider priority assignment policies that are optimal with respect to a specific (sufficient) schedulability test.

Early work on priority assignment

- 1967 Fineberg & Serlin
  - Two periodic tasks with implicit deadlines, better to assign the higher priority to the task with the shorter period
- 1973 Liu & Layland
  - Rate-Monotonic priority ordering is optimal for implicit deadline periodic tasksets (synchronous arrivals)
- 1982 Leung & Whitehead
  - Deadline-Monotonic priority ordering is optimal for constrained deadline tasksets (synchronous arrivals)
  - Deadline Monotonic not optimal for the asynchronous case (offsets)
- 1990 Lehoczky
  - Deadline Monotonic not optimal for arbitrary deadline tasksets
- 1994 Burns et al.
  - Deadline Monotonic not optimal for deadlines prior to completion
- 1996 George
  - Deadline Monotonic not optimal for non-pre-emptive scheduling
Optimality of Deadline Monotonic Priority Assignment

- Deadline Monotonic Priority Ordering (DMPO) **Optimal** for synchronous constrained deadline tasksets
- Response time analysis

\[ R_{i}^{x+1} = C_i + \sum_{\forall j \in hp(i)} \left( \frac{R_{i}^{x}}{T_j} \right) C_j \]

**Proof sketch**

Assume some other priority ordering \( Q \) is schedulable

Swap two tasks \( A \) and \( B \) at adjacent priorities where \( D_A > D_B \) and \( A \) is at a higher priority and the taskset will remain schedulable

Priority order \( Q \): let \( v = R_B \leq D_B \leq T_B \)

Priority order \( P \): \( R_A = v \) (as \( v \leq T_B \)) and so there is interference from only one job of task \( B \), hence as \( D_A > D_B \) task \( A \) is schedulable
Deadline Monotonic: non-optimality

- Tasks with offsets

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Deadline Monotonic: non-optimality

- Tasks with arbitrary deadlines

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>52</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>52</td>
<td>154</td>
<td>140</td>
</tr>
</tbody>
</table>

[Lehoczky J., “Fixed priority scheduling of periodic task sets with arbitrary deadlines”. In proceedings Real-Time Systems Symposium, pages 201–209, 1990]
Deadline Monotonic: non-optimality

- Tasks with deadlines prior to completion

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 + 3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>1 + 0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

[A. Burns, K. Tindell, A.J. Wellings, "Fixed priority scheduling with deadlines prior to completion" In proceedings of the sixth Euromicro Workshop on Real-Time Systems. pp.138-142, 1994]
Non-pre-emptive scheduling

### Table: Task Execution Times

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

The task scheduling diagram illustrates the execution time, deadline, and period for each task. Task A, B, and C are depicted with their respective execution times and deadlines, with a checkmark indicating an optimal scheduling outcome.

Example derived from: [R.I. Davis and A. Burns “Robust priority assignment for messages on Controller Area Network (CAN)”. Real-Time Systems, Volume 41, Issue 2, pages 152-180, February 2009]
Optimal Priority Assignment

for each priority level $i$, lowest first {
  for each unassigned task $\tau$ {
    if $\tau$ is schedulable at priority $i$
    assuming that all unassigned tasks are
    at higher priorities {
      assign task $\tau$ to priority level $i$
      break (exit for loop)
    }
  }
  if no tasks are schedulable at priority $i$ {
    return unschedulable
  }
}
return schedulable

$n(n+1)/2$ schedulability tests rather than $n!$
by exploring all possible orderings
$n = 25$, that is 325 tests rather than 15511210043330985984000000

Optimality of OPA

- Some priority order $Q$ is schedulable
  - Transform $Q$ into the priority order $P$ produced by OPA

- First step
  - Select the task in $Q_n$ that is at priority $n$ in $P$ (Ordering chosen by OPA)
  - Shift that task (from priority $i$) to priority $n$

- $Q_{n-1}$ remains schedulable
  - Higher priority than $i$ - same
  - The task at priority $i$ in $Q_n$ is schedulable at priority $n$ (as it is chosen by OPA)
  - Tasks at priorities $i+1..n$ in $Q_n$ shifted up one in priority - so must remain schedulable
Optimality of OPA (continued)

- At each step $k = n$ down to 1
  - Select the task in $Q_k$ that is at priority $k$ in $P$ (The ordering chosen by OPA)
  - Shift that task (from priority $i$) to $k$
- $Q_{k-1}$ remains schedulable
  - Higher priority than $i$ – same
  - Lower priority than $k$ – same
  - Task at priority $i$ in $Q_k$ is schedulable at priority $k$ as it was chosen by OPA
  - Tasks at priorities $i+1..k$ in $Q_k$ shifted up one in priority – so remain schedulable
- $n-1$ steps to reach ordering $P$ which is schedulable
OPA applicability

OPA algorithm provides optimal priority assignment w.r.t. any schedulability test \( S \) for fixed priority scheduling provided that three conditions are met...

**Condition 1:** Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering

**Condition 2:** Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering

**Condition 3:** When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority

- Tests meeting these conditions referred to as **OPA-compatible**

**Applicability**

- Resource sharing, offsets, arbitrary deadlines, deadlines before completion, non-pre-emptive, CAN, multiframe tasks, mixed criticality, some global FP scheduling on multiprocessors

Multiprocessor: global FP scheduling

- Global FP scheduling
  - Single global run-queue fixed priority pre-emptive scheduling on multiple processors
- Incompatible with OPA ✗
  - Any exact test (B. Andersson and Jonsson 2000) such as those for periodic tasksets given by Cucu and Goossens (2006, 2007).
  - Response time analysis (RTA test) of Bertogna and Cirinei (2007)
  - Improved RTA test of Guan et al. (2009)
- Compatible with OPA ✔
  - Deadline Analysis (DA test) of Bertogna et al. (2009)
  - Simple Response Time test of B. Andersson and Jonsson (2001)

Global FP schedulability tests #1

- Deadline Analysis “DA test” (Bertogna et al. 2009)

\[ D_k \geq C_k + \left\lfloor \frac{1}{m} \sum_{\forall i \in h_p(k)} I_i^D(D_k) \right\rfloor \]

\[ I_i^D(D_k) = \min(W_i^D(D_k), D_k - C_k + 1) \]

\[ W_i^D(D_k) = N_i(D_k)C_i + \min(C_i, D_k + D_i - C_i - N_i(D_k)T_i) \]

\[ N_i(D_k) = \left\lfloor \frac{D_k + D_i - C_i}{T_i} \right\rfloor \]

Compatible with OPA ✓
Global FP schedulability tests #2

- Response Time Analysis “RTA test” (Bertogna & Cirinei 2007)

\[ R_k^{UB} \leftarrow C_k + \frac{1}{m} \sum_{\forall i \in hp(k)} I_i(R_k^{UB}) \]

\[ I_i(R_k^{UB}) = \min(W_i^R(R_k^{UB}), R_k^{UB} - C_k + 1) \]

\[ W_i^R(L) = N_i^R(L)C_i + \min(C_i, L - R_i^{UB} - C_i - N_i^R(L)T_i) \]

\[ N_i^R(L) = \left\lfloor \frac{L - R_i^{UB} - C_i}{T_i} \right\rfloor \]

Incompatible with OPA ✗
Multiprocessor: global FP scheduling

- RTA test dominates DA test
- Which is better?
  - RTA test + heuristic priority assignment
    - Deadline Monotonic
    - D - C Monotonic
    - DkC Monotonic (k is a factor that depends on the number of processors)
  - DA test + Optimal priority assignment

Global FP: Priority Assignment

Percentage of tasksets schedulable

Utilisation

4 Processors
20 tasks

- DA (OPA)
- RTA (DKC)
- DA (DKC)
- RTA (DCMPO)
- DA (DCMPO)
- RTA (DMPO)
- DA (DMPO)
Global FP: Priority Assignment

8 Processors
40 tasks
Global FP: Priority Assignment

16 Processors
80 tasks
Beyond OPA

- What to do if the schedulability test is not OPA-compatible (e.g. RTA test for global FP scheduling)?
  - Search $n!$ combinations?
- How to prune the search space?
  - Use dominance relationship between tests

- Use the sufficient test and the necessary condition to prune the choice of tasks at each priority level

C-RTA necessary test

Based on Response Time Analysis “RTA test” (Bertogna & Cirinei 2007)

\[ R_{kUB}^U \leftarrow C_k + \left[ \frac{1}{m} \sum_{i \in hp(k)} I_i(R_{kUB}^U) \right] \]

\[ I_i(R_{kUB}^U) = \min(W_i^R(R_{kUB}^U), R_{kUB}^U - C_k + 1) \]

\[ W_i^R(L) = N_i^R(L)C_i + \min(C_i, L + R_{kUB}^{UB} - C_i - NN_i^R(LLTT_i)) \]

\[ N_i^R(L) \equiv \left[ \frac{L = R_{kUB}^{UB} - C_i}{TT_i} \right] \]
### Search with backtracking

[Necessary test implies unschedulable]

#### Task Index

<table>
<thead>
<tr>
<th>Priority Level</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>?</td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>x</td>
<td></td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

[OPA-Compatible Sufficient test implies schedulable]

Global FP: Priority Assignment

8 Processors
40 tasks

[3.2 3.6 4.0 4.4 4.8 5.2 5.6 6.0]

Percentage of tasksets schedulable

Utilisation

C-RTA (OPA)
RTA(OPA-Heuristic)
DA-LC (OPA)

Minimising the number of Priority Levels with OPA

Important for practical systems that may support only a limited number of priorities

for each priority level $i$, lowest first {
    $Z = \text{empty set}$
    for each unassigned task $\tau$ {
        if $\tau$ is schedulable at priority $i$ assuming that all unassigned tasks are at higher priorities {
            add $\tau$ to $Z$
        }
    }
    if no tasks are schedulable at priority $i$ {
        return unschedulable
    }
    else {
        assign all tasks in $Z$ to priority $i$
    }
    if no unassigned tasks remain {
        break
    }
} return schedulable

Robust Priority Assignment

- **Drawback of OPA algorithm**
  - Arbitrary choice of schedulable tasks at each priority
  - May leave the system only just schedulable – i.e fragile not robust to minor changes

- **In practice tasks may be subject to additional interference**
  - Execution time budget overruns; interrupts occurring in bursts or at ill-defined rates; ill-defined RTOS overheads; ill-defined critical sections; cycle stealing by peripheral devices (DMA) etc. etc.

- **Want a robust priority ordering**
  - As well as being optimal, able to tolerate the maximum amount of additional interference
  - General model of additional interference \( E(\alpha, w, i) (=\alpha) \)

for each priority level $i$, lowest first
{
    for each unassigned task $\tau$
    {
        determine the largest value of $\alpha$ for which task $\tau$ is schedulable at priority $i$ assuming that all unassigned tasks have higher priorities
    }
    if no tasks are schedulable at priority $i$
    {
        return unschedulable
    }
    else
    {
        assign the schedulable task that tolerates the max $\alpha$ at priority $i$ to priority $i$
    }
}
return schedulable

Ordering achieved in optimal and robust (tolerates the most additional interference (largest $\alpha$) of any priority ordering)
Robust Priority Assignment

- Example: Non-pre-emptive scheduling
  - Additional interference from single invocation of an interrupt handler with unknown execution time
  - Additional interference $E(\alpha, w, i) = \alpha$

<table>
<thead>
<tr>
<th>Task</th>
<th>C</th>
<th>D</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_A$</td>
<td>125</td>
<td>450</td>
<td>450</td>
</tr>
<tr>
<td>$\tau_B$</td>
<td>125</td>
<td>550</td>
<td>550</td>
</tr>
<tr>
<td>$\tau_C$</td>
<td>65</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>$\tau_D$</td>
<td>125</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>$\tau_E$</td>
<td>125</td>
<td>2000</td>
<td>2000</td>
</tr>
</tbody>
</table>
Robust Priority Assignment

- Computed values of $\alpha$

<table>
<thead>
<tr>
<th>Priority</th>
<th>$\tau_A$</th>
<th>$\tau_B$</th>
<th>$\tau_C$</th>
<th>$\tau_D$</th>
<th>$\tau_E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>120</td>
<td>354</td>
</tr>
<tr>
<td>4</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>120</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>110</td>
<td>74</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>135</td>
<td>-</td>
<td>199</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Robust priority ordering
  - Tolerates additional interference of up to 110 time units

- Deadline monotonic: neither optimal nor robust
  - Tolerates additional interference of up to 74 time units

- OPA: may be worse still
  - Might tolerate additional interference of only 10 time units
Robust Priority Assignment

- Mixed systems: two subsets of tasks

  - "DM tasks"
    - Satisfy the restrictions where Deadline Monotonic priority ordering is known to be optimal
    - Pre-emptable, $D \leq T$, resource access according to SRP, no transactions or offsets

  - "Non DM tasks"
    - Don't satisfy the restrictions where Deadline Monotonic priority ordering is known to be optimal
    - Pre-emptable with $D > T$, non-pre-emptable, co-operative scheduling with non-pre-emptable final sections, transactions, non-zero offset

Robust Priority Assignment

- DM task (e.g. constrained deadline)
- Non DM task (e.g. arbitrary deadline, part of a transaction etc.)

For mixed systems containing both DM and non DM tasks, then there exists a robust priority order with the DM tasks in Deadline Monotonic partial order.
Robust Priority Assignment

- Can improve efficiency of OPA and RPA algorithms
  - Of all the DM tasks, the one with the largest deadline is the one that can tolerate the most additional interference at a given priority level
  - Only one DM task need be checked at each priority level - the one with the largest deadline of all unassigned DM tasks
  - For $n$ tasks, $k$ of which are DM tasks:

    $$\frac{n(n+1)-k(k-1)}{2}$$

    task schedulability tests instead of $\frac{n(n+1)}{2}$

- Example: 50 tasks, 4 with $D > T$, 46 constrained deadline tasks
  - max. of 240 schedulability tests instead of 1275

Fixed Priority Scheduling of Mixed Criticality Systems

LO Criticality tasks

DM Priority order

HI Criticality tasks

$2n-1$ schedulability tests rather than $n(n+1)/2$

Fixed Priority Deferred pre-emption Scheduling (FPDS)

- Why limit pre-emptions?
  - Pre-emption costs
    - Context switch time
    - Cache Related Pre-emption Delays (CRPD)
  - Why pre-empt if job is nearly finished
    - May just add overhead for no gain
    - Final non-pre-emptive region

- Task model
  - Each task $\tau_i$ has a final non-pre-emptive region of length $F_i$
  - Increasing $F_i$ improves schedulability of $\tau_i$
  - But decreases schedulability of higher priority tasks via blocking

- FPDS is a superset of:
  - Fixed Priority Pre-emptive Scheduling (FPPS: $F_i = 1$)
  - Fixed Priority Non-pre-emptive Scheduling (FPNS: $F_i = C_i$)
Blocking v. Response Time trade-off

- Task execution

- Blocking

- Response time

- Tolerance of higher priority tasks to blocking: ✗

- Deadline of the task: ✓

Final non-pre-emptive region
Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>175</td>
<td>250</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>325</td>
<td>350</td>
</tr>
</tbody>
</table>

FPPS

For FPPS deadline monotonic is the optimal priority assignment.

FPNS

Trivially not schedulable

100 + 100 > 175
FPDS Priority Assignment

- Deadline Monotonic not optimal

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>175</td>
<td>250</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>325</td>
<td>350</td>
</tr>
</tbody>
</table>

Optimal Fixed Priority Scheduling
with Deferred Pre-emption

Enables optimal FP Deferrable Scheduling
which dominates FPPS and FPNS

Minimises blocking at EVERY priority level

Solution:

for each priority level $i$, lowest first {
    for each unassigned task $\tau$ {
        determine minimum final non-pre-emptable region length 
        (if any) that makes the task schedulable at priority $i$
        assuming that all unassigned tasks have higher priorities
    }
    if no tasks are schedulable at priority $i$ {
        return unschedulable
    } else {
        assign the schedulable task with the shortest final non-
        pre-emptive region at priority $i$ to priority $i$
    }
}
return schedulable

Priority assignment in probabilistic real-time systems

- Tasks with execution times modelled as independent random variables

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Deadline</th>
<th>Period</th>
<th>Permitted DMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(2, 3)</td>
<td>5</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>(0.7, 0.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(3, 4)</td>
<td>6</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>(0.8, 0.2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Deadline monotonic priority ordering not optimal
  - Task A at higher priority $P(R_A > D_A) = 0 \checkmark$  $P(R_B > D_B) = 0.06 \times$
  - Task B at higher priority $P(R_B > D_B) = 0 \checkmark$  $P(R_A > D_A) = 0.44 \checkmark$

Optimal Priority Assignment for probabilistic systems

Same three conditions needed for OPA compatibility

- Condition 1: Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering.
- Condition 2: Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering.
- Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority.

Definition of “schedulable” very different – based on probability of deadline failure (i.e. response time distribution and its exceedance function) compared to permitted Dead Miss Ratio.
Interesting problems not obviously amenable to OPA algorithm

- **Probabilistic:**
  - Minimising average/total probability of deadline failure across all tasks (Maxim et al 2011)
    - Swapping tasks at adjacent priorities may decrease the total, even if the larger of the two probabilities of deadline failure decreases

- **FPDS:** Minimising the number of pre-emption through maximising blocking (Bertogna et al 2011)
  - Can be done from highest priority down rather than lowest priority up, but then requires a pre-defined priority ordering

- **NoC wormhole communication:** Assigning priorities to network flows (Shi and Burns, 2008)
  - Response time of a network flow depends on the response times of higher priority flows

- **Pre-emption thresholds:** Assignment of base priorities and pre-emption thresholds (Wang and Saksena, 1999)
  - Pre-emption threshold assignment depends on the relative priority ordering of higher priority tasks
Interesting problems not obviously amenable to OPA algorithm

- **Cache Related Pre-emption Delays (CRPD)**
  - Response times depend upon the relative priority ordering of higher priority tasks

---


[S. Altmeyer, R.I. Davis, C. Maiza “Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems”. In proceedings 32nd IEEE Real-Time Systems Symposium (RTSS’11), pages 261-271, Nov 29th - Dec 2nd, 2011]
Beyond the OPA algorithm: RTSOPS 2013 Open Problem

- **OPA-incompatible analysis**
  - Can theoretically find an optimal priority assignment by exploring all \( n! \) possibilities – but this is not tractable

- **Open problem:** (posed by Alan Burns at RTSOPS 2013)
  - With uniform hardware and run-time support, if the three OPA-compatibility conditions do not apply, can optimal priority assignment only be achieved via exhaustive enumeration of all priority orderings?

| Condition 1: Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering |
| Condition 2: Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering |
| Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority |

- Are there scheduling problems and analysis where something between \( > O(n^2) \) and \( < O(n!) \) complexity can provide an optimal solution?
Beyond OPA algorithm: RTSOPS 2013 Open Problem

- Contrived example: Hardware accelerator for the single highest priority task (non-uniform hardware)
  - Task at the highest priority executes in half the time
  - Analysis now breaks the conditions for OPA-compatibility

  Condition 1: Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering
  Condition 2: Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering
  Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority

- Solution: try each task in turn at the highest priority and apply OPA to the remaining priority levels: $O(n^3)$

- Are there practical and useful problems amenable to priority assignment of polynomial complexity (in the number of single task schedulability tests) where we can’t use OPA algorithm?
Finally… a 20 year old Open Problem

- **Dual Priority scheduling**
  - Each task has two priorities
  - A fixed time $S_i$ after task $\tau_i$ is released, its priority is promoted to the higher of its two priorities

- **Hypothesis**
  - Utilisation bound for Dual Priority scheduling of implicit deadline sporadic tasks is 100%
  - Proved for $n = 2$
  - No counter examples found so far for $n > 2$

- **Problem:** Choosing priorities and promotion times

[A. Burns, “Dual Priority Scheduling: Is the Processor Utilisation bound 100%” In proceedings RTSOPS, 2010.]
Questions?