Scheduling

Introduction
Distinguishing characteristic of real time systems
Have specified time constraints that must be met
Hard or soft constraints
To be able to meet hard real time deadlines
Must be able to determine prior to deployment
Whether or not deadlines can be met
Certainly one approach
Consider and schedule according to worst-case behaviour
While such an approach
Ensures that constraints can be met
Yields relatively poor performance

Have seen several general classes of schedule

• Static
  Schedule determined and set before deployment
  Typically implemented as units of
  Least common multiple of task durations
  Major problem once again inefficiency
  Typically does not need an OS
  Can easily implement with simple task queue
  Most effective when number of tasks fixed and
  Subject to very little modification
  To be effective
  Knowledge of worst-case times
  Essential

• Priority
  Tasks assigned a priority
  Although not required
  Such systems generally use RTOS
  Assumed that RTOS supports pre-emption
  RTOS ensures
  Highest priority task running at any time
  Has potential of being more efficient than static scheme
  Difficulty
  Worst-case response times not immediately obvious
  Individual tasks
  Interrupts and associated handler
Without such knowledge
  becomes very difficult to try to use priority scheme

Can develop such understanding
  Using approach called
    Deadline Monotonic Analysis
  Scheme provides means to
    Analyze how tasks in priority-based schedule interact
    Ascertain worst-case response times
      Tasks
      Interrupts and interrupt handler

Deadline Monotonic Analysis
  To work through deadline monotonic analysis

Assumptions
  We establish some simplifying assumptions to begin with
    Will relax several as we develop the analysis

  1. We have a fixed set of tasks with known priorities
  2. Any tasks can be ready to run at any time
    Known minimum interval before can be ready again
  3. Execution duration is bounded
  4. Task cannot suspend itself
    Cannot wait or block on event
  5. Task deadline must be less that or equal to its period
    Related to point 2 above
  6. Duplicate priorities not allowed
  7. Tasks are preemptable and cannot be delayed by lower priority tasks
    Under such conditions tasks cannot
      Disable interrupts
      Share data via a semaphore
  8. Time to schedule and execute context switch known and bounded

As we proceed
  Will examine and modify last two assumptions

Vocabulary
  Let’s now introduce some the necessary vocabulary

  - $T_i$ – Period of task $i$
Time between successive instances of task ready to run

- \( C_i \) - Execution time on system processor for task \( i \)
  This is strictly task execution time of the \( i \)th task
  Does not include time for any nontask processing
  Other tasks or interrupts

- \( R_i \) - Response time of task \( i \)
  Time between task
  Becoming ready to run and
  Completing worst case execution time
  Note this time will include time for any nontask processing
  Other tasks or interrupts

- \( I_i \) - Time for other tasks and interrupts during task \( i \)

- \( D_i \) - Deadline for task \( i \)
  Time when task must be completed
  Clearly task will always meet deadline if
  \( D_i \geq R_i \)

Analysis
From above definitions
Following relationship is clear
1. \( R_i = C_i + I_i \)

Presuming that we know task execution time \( C_i \)
Problem reduces to finding \( I_i \)

Given a second task \( j \)
Number of times, \( N \), it can preempt executing task \( i \) is given by
Ceiling of ratio of response time for task \( i \) and period of task \( j \)

Thus
2. \( N = \left\lceil \frac{R_i}{T_j} \right\rceil \)

For given \( x \)
Ceiling function returns smallest integer \( \geq x \)

From which we compute the total time consumed by second task

3. \( \text{time} = \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \)
From eq 3 can easily compute time consumed for all tasks
Simply sum value over all higher priority tasks, $T_{hp}$

\[ I_i = \sum_{k \in T_{hp}} \left[ \frac{R_i}{T_j} \right] \cdot C_k \]

Substituting back into 1 above we have following

\[ R_i = C_i + \sum_{k \in T_{hp}} \left[ \frac{R_i}{T_j} \right] \cdot C_k \]

Which leads us to the recurrence relationship

\[ R_i^{n+1} = C_i + \sum_{k \in T_{hp}} \left[ \frac{R_i^n}{T_j} \right] \cdot C_k \]

Now extend eq 5 to relax restriction of no blocking
To accommodate case when task can be blocked
By lower priority task
Denoted *priority inversion*

\[ R_i = B_i + C_i + \sum_{k \in T_{hp}} \left[ \frac{R_i}{T_j} \right] \cdot C_k \]

Term $B_i$ expresses time when such blocking occurs
Includes time for all blocking tasks

Computing value of $B_i$ can be difficult
Placing restriction on such blocking
Makes problem tractable
Restriction
Higher priority task blocked at most once
By all lower priority tasks
Such restriction called
*Priority Ceiling Protocol*
**Priority Ceiling Protocol**

Stipulates that each semaphore has ceiling priority
- Priority of highest priority task that can lock semaphore
- Task can simultaneously hold multiple semaphores
- Must be locked and unlocked in nested pattern

Lock 1
- Lock 2
- Lock 3
- Unlock 3
- Unlock 2
- Unlock 1

Using *instant inheritance* algorithm
- When task
  - Locks semaphore
    - Priority of task raised to ceiling priority of semaphore
  - Unlocks semaphore
    - Priority restored to original value
- Consequence lock always succeeds
  - No other task can have semaphore locked
  - If so
    - Such task would have ceiling priority and be running
    - Current task could not be running

Based upon such knowledge
- Can now compute blocking time for task $i$
- First we define $T_{lp}$
  - Set of lower priority tasks than task $i$

Examine set of semaphores that can be locked by each task $j$ in $T_{lp}$
- Select subset of all such semaphores
  - Have ceiling priority higher than that of task $i$
  - Define such a subset as $S_j$

Define $Time_{j,s}$ as time task $j$ holds semaphore $s_k$ in $S_j$

We now have

$$B_i = \max_{\forall j \in T_{lp}, \forall s_k \in S_j} \left( Time_{j,s} \right)$$

Let’s now look at including overhead for
Scheduler and context switch

We recognize scheduler running continuously

Thus we have two components to time burden
1. Time to execute basic scheduler when no tasks need to be handled
2. Time to manage actual context switch

For a given duration and task with period $T_k$

Scheduler can be summoned number of times given by

$$9. \text{ invocations} = \frac{t}{T_k}$$

Time burden for single task then given by

$$10. \text{ time} = \left\lfloor \frac{t}{T_k} \right\rfloor \cdot C_{\text{task}k}$$

That for all tasks follows simply

$$11. \text{ time}_{\text{total}} = \sum_{\forall \text{tasks} \in \text{total ScheduledTasks}} \left\lfloor \frac{t}{T_k} \right\rfloor \cdot C_{\text{task}k}$$

Time to execute basic scheduler for same duration

$$12. \text{ time}_{\text{scheduleBase}} = \left\lfloor \frac{t}{T_{\text{schedule}}} \right\rfloor \cdot C_{\text{schedule}}$$

In this case we have

Period of the scheduler – $T_{\text{schedule}}$
Worst-case execution time of scheduler – $C_{\text{schedule}}$

Combining we now have

$$13. \text{ time}_{\text{schedule}} = S_i = \left\lfloor \frac{t}{T_{\text{schedule}}} \right\rfloor \cdot C_{\text{schedule}} + \sum_{\forall \text{tasks} \in \text{total ScheduledTasks}} \left\lfloor \frac{t}{T_k} \right\rfloor \cdot C_{\text{task}k}$$

Adding the time burden for scheduler overhead to our response time for task $i$

We have

$$14. R_i = S_i + B_i + C_i + \sum_{k \in T_{wp}} \left\lfloor \frac{R_i}{T_j} \right\rfloor \cdot C_k$$

The final piece of the equation includes

Time burden of context switch, $C_{\text{switch}}$
Burden comprises several components
1. Time to suspend and save task i
2. Time to restore and resume task i
3. Time to activate preempting tasks
4. Time to suspend and save preempting tasks

First two items
Included once

Latter two items must be included for each possible preempting task

Adding this final component we now have

\[ R_i = S_i + B_i + C_i + C_{\text{switch}} + \sum_{k \in T_j} \left( \frac{R_i}{T_j} \right) \bullet (C_k + C_{\text{switch}}) \]

In the equation above
First instance of \( C_{\text{switch}} \)
Covers items 1 and 2
Second instance
Covers items 3 and 4