Rate Monotonic Scheduling

Reference:

http://www.netrino.com/Publications/Glossary/RMA.php

Fall 2007
Rate Monotonic Scheduling

• The scheduling algorithm you choose depends on your goals.
• Different algorithms yield different results.
• Suppose you're given 10 jobs and each will take a day to finish.
• In ten days, you will have all of them done.
• But what if one or more has a **deadline**?
• If the ninth task given to you has a deadline in three days,
• Doing the tasks in the order you receive them will cause you to miss that deadline
• The purpose of a real-time scheduling algorithm is to ensure that critical timing constraints, such as deadlines and response time, are met.

• Real-time scheduling is also used to allocate processor time between tasks in soft real-time embedded systems.

• Many real-time systems use preemptive multitasking, especially those with an underlying real-time operating system (RTOS).

• Priorities are assigned to tasks, and the RTOS always executes the ready task with highest priority.
• The scheduling algorithm is the method in which priorities are assigned.

• Most algorithms are classified as static priority, dynamic priority, or mixed priority.

• A static-priority algorithm assigns all priorities at design time, and those priorities remain constant for the lifetime of the task.

• A dynamic-priority algorithm assigns priorities at runtime, based on execution parameters of tasks, such as upcoming deadlines.

• A mixed-priority algorithm has both static and dynamic components.
Static-priority algorithms are simpler than algorithms that must compute priorities on the fly.

The rate monotonic algorithm (RMA) is a procedure for assigning fixed priorities to tasks to maximize their "schedulability."

A task set is considered schedulable if all tasks meet all deadlines all the time.

The algorithm is:

“Assign the priority of each task according to its period, so that the shorter the period the higher the priority”
• RMA is an optimal static-priority algorithm.

• If a task set cannot be scheduled using the RMA algorithm, it cannot be scheduled using any static-priority algorithm.

• One major limitation of fixed-priority scheduling is that it is not always possible to fully utilize the CPU.

• Even though RMA is the optimal fixed-priority scheme, it has a \textit{worst-case schedule bound} of:

\[ U = n \left[ 2^{(1/n)} - 1 \right] \]

where \( n \) is the number of tasks in a system.
\[ U = n \left[ 2^{(1/n)} - 1 \right] \]

Note that the worst-case schedulable bound for one task is 100%.

As the number of tasks increases, the schedulable bound decreases, eventually approaching its limit of about 69.3% (\( \ln 2 \), to be precise).

To probe this value \( \ln 2 \), we need to recall l'Hopital’s Rule.
l’Hopital’s Rule uses derivatives to help compute limits with indeterminate forms.

l’Hopital’s rule state that for function \( f(x) \) and \( g(x) \) if

\[
\lim_{x \to c} f(x) = \lim_{x \to c} g(x) = \pm \infty,
\]

then:

\[
\lim_{x \to c} \frac{f(x)}{g(x)} = \lim_{x \to c} \frac{f'(x)}{g'(x)}
\]

where the \( ' \) denotes the derivative
Theorem (RMA Bound) Any set of \( n \) periodic tasks is RM schedulable if the processor utilization, \( U \), is no greater than \( n(2^{1/n} - 1) \).

This means that whenever \( U \) is at or below the given utilization bound, a schedule can be constructed with RM. In the limit when the number of tasks \( n = \infty \), the maximum utilization limit is

\[
\lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 \approx 0.69
\]

The calculation of the limit in Equation 3.7 is straightforward but worth documenting. First recall that

\[
\frac{d}{dx} a^x = (\ln a)a^x dx
\]

Hence,

\[
\frac{d}{dn} 2^{n^{-1}} = (\ln 2)2^{n^{-1}} (-n^{-2})
\]

Now

\[
\lim_{n \to \infty} n(2^{n^{-1}} - 1) = \lim_{n \to \infty} \frac{(2^{n^{-1}} - 1)}{n^{-1}}
\]

And, by L’Hôpital’s rule

\[
\lim_{n \to \infty} \frac{(2^{n^{-1}} - 1)}{n^{-1}} = \lim_{n \to \infty} \frac{\frac{d}{dn} (2^{n^{-1}} - 1)}{\frac{d}{dn} (n^{-1})}
\]

\[
\lim_{n \to \infty} \frac{\frac{d}{dn} (2^{n^{-1}} - 1)}{\frac{d}{dn} (n^{-1})} = \lim_{n \to \infty} \frac{\ln 2(2^{n^{-1}})(-n^{-2})}{-n^{-2}} = \lim_{n \to \infty} \frac{\ln 2(2^{1/n})}{1} = \ln 2
\]
Guidelines

To benefit most from using a fixed-priority preemptive RTOS, consider the following rules of thumb:

• Always assign priorities according to RMA.

• If total utilization is less than or equal to $U(n)$, all tasks will meet all deadlines, so no additional work needs to be done.

• If total utilization is greater than $U(n)$, an analysis of the specific task set is needed, to verify whether or not it will be schedulable.

• To achieve 100% utilization when using fixed priorities, assign periods so that all tasks are harmonic.
• Harmonic Tasks means that for each task, its period is an exact multiple of every other task that has a shorter period.

• The last rule of thumb provides a simple guideline for most efficient use of the processor:

For example, a three-task set whose periods are 10, 20, and 40, respectively, is harmonic, and preferred over a task set with periods 10, 20, and 50.
Extended Rate Monotonic Scheduling
Extended Rate Monotonic Scheduling

Now we have to consider blocking of resources due to resource contention

\[
\frac{C_1}{p_1} + \ldots + \frac{C_i}{p_i} + \frac{B_i}{p_i} \leq U(i) = i \left(2^{1/i} - 1\right)
\]

\(1 \leq i \leq n\)

\(B_i = \text{longest duration of blocking that can be experienced by } i\)

\(C_i = \text{worse case execution time associated with periodic task } i\)

\(p_i = \text{period associated with task } i\)

\(n = \text{number of tasks}\)
The following diagram shows 3 tasks with their respective worse case execution times and periods. There are also two resources, with their respective duration for each task. Find the extended RMS equations for each resource. Is there resource contention for this example?. Explain.

\[
\frac{C_1}{p_1} + \ldots + \frac{C_i}{p_i} + \frac{B_i}{p_i} \leq U(i) = i \left( 2^{\frac{1}{i}} - 1 \right)
\]

\[1 \leq i \leq n\]
Solution:

- **Resource1**: this resource doesn’t have any issue since only one task is using it.

- **Resource2**: looking at its schedulability, yields three separate equations

  1.- For Task1: \( i = 1 \)
  
  \[
  \frac{C_1}{p_1} + \frac{B_1}{p_1} = \frac{20}{100} + \frac{18}{100} = 0.38 \\
  \text{U}(1) = 1 \left( 2^{1/1} - 1 \right) = 1
  \]

  It is true that \( 0.38 \leq 1 \)

  Therefore there is no resource contention for Resource2 from Task1
2.- For Task2: \( i = 2 \)
\[
\frac{C_1}{p_1} + \frac{C_2}{p_2} + \frac{B_2}{p_2} = \frac{20}{100} + \frac{30}{150} + \frac{18}{150} = 0.52
\]
\[
U(2) = 2\left(2^{\frac{1}{2}} - 1\right) = 0.8284
\]
It is true that \( 0.52 \leq 0.8284 \)

Therefore there is no resource contention for Resource2 from Task2

3.- For Task3: \( i = 3 \)
Because Task3 is low priority, Resource2 can always be preempted by other task, therefore \( B_3 = 0 \)
\[
\frac{C_1}{p_1} + \frac{C_2}{p_2} + \frac{C_3}{p_3} = \frac{20}{100} + \frac{30}{150} + \frac{50}{300} = 0.567
\]
\[
U(3) = 3\left(2^{\frac{1}{3}} - 1\right) = 0.779
\]
It is true that \( 0.567 \leq 0.779 \)

Therefore there is no resource contention for Resource 2 from Task3
Dynamic Priority Algorithm
Dynamic Priority Algorithm

- Dynamic Priority Algorithm
  - Earliest deadline first (EDF) scheduling
  - Example
    - Consider 3 periodic processes scheduled using EDF, the following acceptance test shows that all deadlines will be met.
    - The theoretical limit for any number of processes is 100% and so the system is schedulable.
      - \( \frac{e_i}{p_i} \leq 1 \)

<table>
<thead>
<tr>
<th>Process</th>
<th>Execution Time</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

\[
\frac{1}{8} + \frac{2}{5} + \frac{4}{10} = 0.925
\]
What happens if not scheduled properly

• Problems:
  ▪ Priority Inversion
  ▪ Missed Deadlines
  ▪ Deadlock
Priority Inversion

• Unwanted software situation when high-priority task is delayed while waiting for shared resource that is not in use.

• Priority inversion arises when a medium-priority task preempts lower priority task using a shared resource on which the higher priority task is pending.

• If higher priority task is otherwise ready to run, but a medium-priority task is currently running instead, a priority inversion is said to occur.
Deadline

• Time at which a particular set of computations or data transfers must be completed

• Considered “Hard Deadline” if it ABSOLUTELY MUST be met every time

• Hard Deadlines have serious consequences when missed

• Other Deadlines are said to be “Soft Deadlines”
Deadlock

• Entire set of tasks is blocked, waiting for an event that only a task within the same set can cause

• Only solution is to reset the involved set of tasks or entire system

• Usually possible to prevent deadlocks by following certain software-design practices
Deadlock
Avoiding Deadlock

- Acquire all resources before proceeding
- Acquire resources in same order
- Release all resources in reverse order
- A timeout in a semaphore (provided by most kernels) can be a way out of deadlock
A shared resource is a resource that can be used by more than one task.

Each task should gain exclusive access to the shared resource to prevent data corruption.

This process is called Mutual Exclusion.

Exist techniques that ensures the Mutual Exclusion.
Mutual Exclusion (cont.)

- Easiest way to communicate is through shared data such as
  - Global variables
  - Pointers
  - Buffers
  - Linked lists
  - Ring buffers

- However this could cause data corruption
Mutual Exclusion (cont.)

- Sharing data simplifies exchange of information

- Care must be take to ensure that each task has exclusive access to the data to avoid contention and data corruption

- Common 4 methods of gaining exclusive access to shared resources:
  1. Disabling interrupts
  2. Performing test-and-set operations
  3. Disabling scheduling, and
  4. Using semaphores
1. Disabling and enabling interrupts

- Example
  - Disable interrupts
  - Access the resource (read/write from/to variable)
  - Enable interrupts

MicroC/OS-II provides macros for this

OS_ENTER_CRITICAL()
OS_EXIT_CRITICAL()
Using μC/OS-II macros to disable and enable interrupts

Void Function (void)
{
    
    OS_ENTER_CRITICAL();
    . /* you can access shared data in here */
    .
    OS_EXIT_CRITICAL();
}

2. **Test-and-Set Operation**

- When using a kernel
  - Two functions could agree to access a resource
  - They check global variable for 0 to gain access
  - To prevent other function from accessing the resource, first function that reaches the resource sets this variable to 1
- This is referred to as *test and set (TAS)*
3. Disabling and Enabling the Scheduler ("Locking and Unlocking the Scheduler")

- If a task is not sharing variables or data structures with an ISR, scheduling can be disabled and enabled.
- Using this method, two or more tasks can share data without contention.
- This method works very well, however should be avoided because disabling the scheduler defeats the purpose of having a kernel.

A much better option is the use of Semaphores.

ISR - interrupt service routine
4. Semaphores

• Key that your code acquires in order to continue execution

• If semaphore already in use, the requesting task is suspended until the semaphore is released by its current owner

• Two types of semaphores:
  • Binary
    • Can take only two values 0 or 1
  • Counting
    0 to 256 (8 bit)
    0 to 65,535 (16 bit)
    0 to 4,294,967,295 (32 bit)