

ESAII-RR-00-17

Juliol 2000

Rate Monotonic Scheduling Timing Analysis

Pau Martí, Josep M. Fuertes and Gerhard Fohler

Index

Executive summary	3
1. Real time systems basic concepts	4
2. Real time scheduling	6
3. Real start time timing analysis	7
3.1. Example	7
3.2. Algorithm description	8
3.3. Procedures	9
4. Conclusions	12
References	12

Executive summary

In real time systems, scheduling theory addresses, by means of algorithms, the problem of meeting the specified timing requirements in order to have an understandable, predictable and maintainable system timing behaviour. One of the most commonly used scheduling algorithms is Rate Monotonic Algorithm. This offline fixed priority algorithm produces schedules where the start time of a given task instance varies from task instance execution to task instance execution. In this report we present a set of procedures to calculate the real starting time of a task instance, given a set of schedulable tasks under Rate Monotonic Algorithm.

1. Real time systems basic concepts

Real time computing has been widely used in many areas during the last three decades, playing a key role in technology. There are many definitions of real time computing, but basically, the main idea suggested by Stankovic (1988) is that in such a computing system, the correctness of the system depends not only on the logical results of the computations but also on the time at which the results are produced.

In real time systems, scheduling theory addresses, by means of algorithms, the problem of meeting the specified timing requirements in order to have an understandable, predictable and maintainable system timing behaviour. Therefore, as said by Ramamritham and Stankovic (1994), scheduling involves the allocation of resources and time in such a way that certain performance requirements are met.

Real time systems are characterized by a set of concurrently executing tasks that have deadlines, which represent the time at which the task should complete its execution so as not to cause damage to the system.

Depending on the consequences of a missed deadline (deadline miss tolerance), two types of real time systems can be differentiated:

- Hard real time systems are those where it is imperative that responses occur within the specified deadline
- Soft real time systems are those where response times are important but the system will still function if deadlines are occasionally missed.

In fact, it has to be pointed out that there is no complete agreement on the previous definitions. Sometimes, if the consequences of missing a deadline are catastrophic, the target system is called critical (or safety critical) real time system instead of hard real time system.

Another timing characteristic that can be specified on a real time task concerns the regularity of its activation. Depending on it, a task is defined as a periodic or aperiodic task. A task that is released at regular intervals is a periodic task. On the other hand, a task that is not invoked at regular intervals is an aperiodic task. To allow worst-case calculations to be made there is often defined a minimum period between any two aperiodic events (from the same source). If this is the case, the task involved is said to be sporadic.

In general, a real time task τ_i can be characterised by several of the parameters described in table 1, which can be directly task attributes or attributes obtained from current scheduling methods.

Table 1. Parameters for a real time task τ_i

Parameter	Description
Criticalness (hard/soft)	Parameter related to the consequences of missing deadlines
Worst case computation time (C_i)	An estimation of the maximum time required by the processor for executing the task without interruption
Period (T_i)	Time between consecutive tasks releases
Deadline(D_i)	Time before which the task should be completed
Priority (P_i)	Relative importance (for scheduling purposes) of the task with respect to the other tasks in the system
Starting time (s_i)	Time at which a task starts its execution
Finishing time (f_i)	Time at which a tasks finishes its execution
Worst case blocking time (B_i)	Maximum time during which the task may be blocked by lower priority tasks when accessing a resource
Interference (I_i)	Maximum time that the task will have to wait due to preemption from higher priority tasks
Processor utilization (U_i)	Processor utilization of the task (equal to C_i/T_i)
Worst case response time(R_i)	Longest time duration between the invocation of a task and the time that the task completes
Release time (r_i)	Time at which a task becomes ready for its execution
Jitter (J_i)	Time that a task spends from its release until its start time
Lateness (L_i)	Difference between a task completion time and its deadline
Tardiness (E_i)	Time during which a task stays active after its deadline
Laxity (slack time) (X_i)	Maximum time a task can be delayed on its activation to complete within its deadline

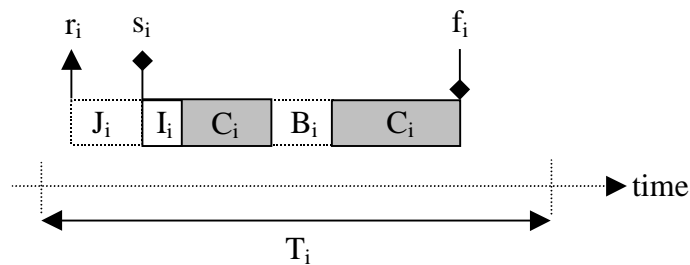
Figure 1. Some parameters for a real time task τ_i

Figure 1 shows some of the parameters of a real time task. To characterise all tasks in any distributed real time system, the above parameters should be combined with other models in order to specify resource constraints, concurrency relations, precedence relations, communication patterns and placement constraints that any set of real time tasks may have.

2. Real time scheduling

Since the early work by Liu and Layland (1973) on real time scheduling, many scheduling algorithms and methods have been presented.

In Tindell and Hansson (1997), the scheduling approach is broken down into three main activities: off-line configuration, run-time dispatching, and a priori analysis. The off-line configuration generates the static information that will be used for the run-time dispatching. The run-time dispatching deals with the switching between computations for different events at run-time. The a priori analysis examines the system and tells us if all the timing requirements will be met according to the off-line configuration information and the behaviour of the run-time dispatching algorithm. Scheduling approaches can differ on how much effort they spend in every activity.

The a priori analysis is used to determine by means of tests the feasibility of scheduling approaches, i.e. whether the temporal constraints of all processes will be met at run time. The analysis is said to be sufficient and necessary when a task set will meet all its deadlines if, and only if, it passes the test. If passing the test means that the task will meet all its deadlines at run time, but failing the test does not imply that the task will not meet all its deadlines, the analysis is said to be sufficient and not necessary, as Audsley, *et al.* (1995) showed.

Among the great variety of real time scheduling approaches, four main classes are identified by Ramamritham and Stankovic (1994): off-line vs. on-line, preemptive vs. non-preemptive, static vs. dynamic and optimal vs. heuristic scheduling. Obviously, any scheduling algorithms may belong to more than one of the identified classes.

In this report we focus on off-line scheduling, concretely, in the Rate Monotonic Algorithm presented by Liu and Layland (1973). This fixed priority algorithm is based on the following priority assignment: a task with a shorter period should be assigned a higher priority. At run time then, the dispatcher will make sure that at any time, the highest priority runnable task is actually running. That means that if we have a task with a low priority running, and a high priority task arrives, the low priority task will be suspended, and the high priority task will start running (preemption). Table 2 illustrate the RM priority assignment (1 is the highest priority, 3 is the lower).

Table 2. Priority assignment

Task	τ_1	τ_2	τ_3
Period	5	14	18
Priority	1	2	3

Therefore, applying RMA on any given set of tasks, before run time, the offline schedule can be obtained. In these schedules its easy to see that the starting time of each instance of each task is not always the theoretical starting time that could have been supposed.

That is, for the k -instance of the task i with period T_i , the starting time is not always $k \cdot T_i$, its theoretical start time. This phenomena is due to the fact that the target instance has to delay its execution start because other higher priority tasks are executing.

What we have developed is a set of procedures that allow to calculate offline, the real starting time of the k -instance of the task i given a set of schedulable *tasks* under RMA. This procedures have been implemented in Matlab and integrated into the Real Time Control Systems Simulator presented by J. Eker and A. Cervin (1999).

3. Real start time timing analysis

In this section, the real starting time calculation of the k -instance of the task i given a set of schedulable *tasks* under RMA is presented. Firstly, using an example, we explain why the theoretical and real start times are different. Afterwards, we describe the algorithm and finally we present all the implemented procedures.

3.1. Example

To illustrate the problem and the way the algorithm works, we will use the task set specified in table 2.

Table 3. Task set

	τ_1	τ_2	τ_3
T_i	5	14	18
C_i	2	4	2

The following scheme (figure 2) shows a partial schedule for the given set of schedulable tasks under Rate Monotonic scheduling algorithm. It can be seen that the real starting time of each instance of each task is not always the theoretical starting time of the instance ($k \cdot \tau_i$). For example the real starting time of the 1 -instance of the task 3 is not 0.18 t.u. ($1 \cdot 0.18$), it is 0.22 t.u.

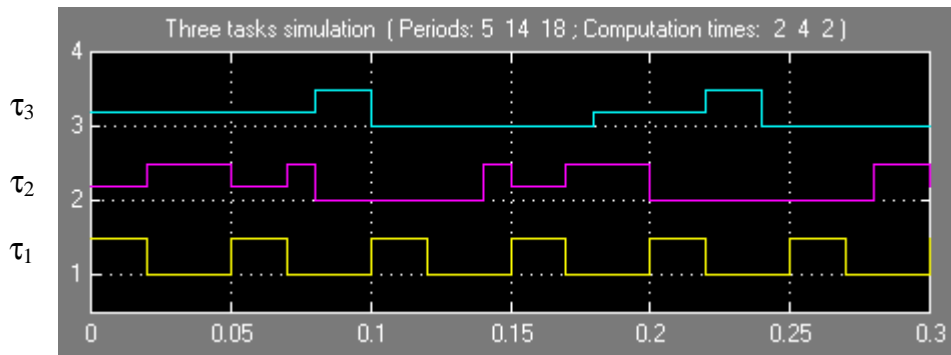


Figure 2. Partial schedule

(Notice that in figure 2, for each task execution, there are three levels. The lower level means that the task doesn't have to execute, the upper level means that the task is executing, and the middle level means that the task has to execute, but it is delayed because other higher priority tasks are executing)

3.2. Algorithm description

The following algorithm calculates the real starting time of the k -instance of the task i given a set of schedulable *Tasks* under RM algorithm. The basic idea is, given the theoretical starting time of the k instance of the task i ($k \cdot T_i$), the following steps must be performed:

- Calculate the initial time instant (iti) from which the real starting time of the target instance can be delayed
- From the initial time instant, calculate the delay period (dp= $ip+es$) which is the interference period (ip) plus the empty slots (es) that are actually delaying the real starting time of the task instance.
- The interference period is the execution time of all higher priority tasks than the target task within a given period if and only if, their execution can delay the starting time of instance.
- The empty slots is the remaining time that is not used for execution of tasks but it can also delay the starting time of the target instance
- The real start time will be the initial time instant plus the delay period.

With the presented task set, and taking into account the schedule of figure 2, a numerical trace of the schedule can be stated as follows:

Theoretical starting time of the second instance of task τ_3 ($kT_i, k=1$) = 0.18
 Initial time instant = 0.10
 Delay period = Interference period + Empty slots = 0.10+0.2 = 0.12
 Real starting time = Initial time instant + delay period = 0.10 + 0.12 = 0.22

Therefore, the algorithm can be understood (figure 3), from the theoretical starting time (TST) of the k -instance of the task i as a backward (calculate the initial time instant, ITI) - forward (calculate the delay period, DP) process in order to obtain the real starting time (RST).

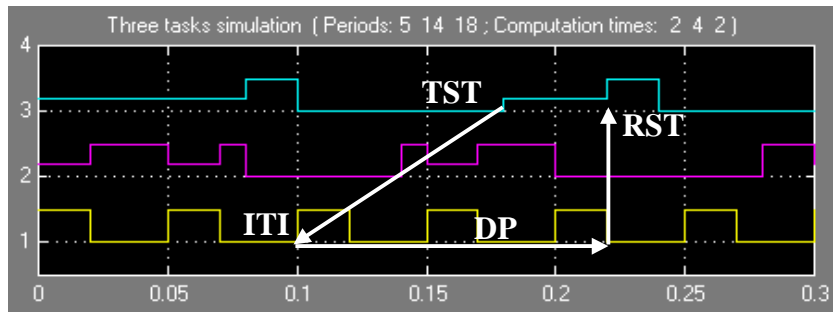


Figure 3. Algorithm scheme

3.3. Procedures

In this section, we'll review all the procedures that have been implemented. However, for the sake of clarity, we will fix the notation that we will use in the explanation of each procedure.

Notation

Tasks (ordered by priority): $\tau_1, \tau_2, \dots, \tau_n$

Period of τ_i : T_i

Computation time of τ_i : C_i

Priority of τ_i : i

K-instance: k (0,1...)

Time Unit (t.u.): 1

Procedure 1: Time Instant

The first step in the algorithm is to calculate the theoretical starting time the k -instance of the task i , which is $k \cdot T_i$

The time instant from which any task j of higher priority than task i can start to delay the real starting time of the k -instance of the task i is:

$$\boxed{TI_i^k(j) :} \quad TI_i^k(p) = \left\lfloor \frac{k \cdot T_i}{T_p} \right\rfloor \cdot T_p$$

$$\text{For } (p = i-1 : -1 : j) \quad TI_i^k(p-1) = \left\lfloor \frac{TI_i^k(p)}{T_{p-1}} \right\rfloor \cdot T_{p-1}$$

Implementation:

```
function ti=TI(j,k,i,Tasks)
    ti=k*Tasks(1,i);
    for p=i-1:-1:j
        oldti=ti;
        ti=fix(oldti/Tasks(1,p))*Tasks(1,p);
    end
```

(this procedure can also be used to calculate the initial time instant from which the k -instance of the task i can be delayed due to all higher priority tasks than i in a given set of tasks. In this case, j will be I)

Procedure 2: Interference Period

The new interference period from a given initial period that can delay the starting time of the k -instance of the task i due to execution of higher priority tasks j than task i , if and only if, their execution can delay its start time, is:

$$IP_i^k(t)^{n+1} = \sum_{j: I_j \leq I_i + t} \left[\frac{IP_i^k(t)^n - (TI_i^k(j) - TI_i^k(1))}{T_j} \right] \cdot C_j$$

$$\text{Starting at} \quad IP_i^k(t)^0 = t$$

$$\text{Finishing when} \quad IP_i^k(t)^{n+1} = IP_i^k(t)^n$$

Implementation:

```
function ip=IP(t,k,i,Tasks)

    oldip=0;
    ip=t;

    while (ip>oldip)

        oldip=ip;
        ip=0;

        for j=1:i-1

            if TI(j,k,i,Tasks)<=TI(1,k,i,Tasks)+t

                ip= ip + up(oldip+TI(1,k,i,Tasks)-CI(j,k,i,Tasks),Tasks(1,j))
                    *Tasks(2,j);

            end

        end

    end

end
```

Procedure 3: Empty Slots

The empty slots (time units, ticksize) from a given initial period t that can delay the starting time of the k -instance of the task i due to remaining not used time is:

$$\boxed{ES_i^k(t) :} \quad es = 0$$

$$\text{For } (p = 1 : t) \quad es = es + \max(0, p - (IP_i^k(p)^n + es))$$

Implementation:

```
function es=ES(t,k,i,Tasks,ticksiz)
    es=0;
    for p=ticksiz:ticksiz:t
        es=es+max(0,p-(IP(p,k,i,Tasks)+es));
    end
```

Procedure 4: Real Start Time

The final algorithm could be represented by the following steps:

1.- To find a delay period, $dp=ip+es$:

$$RST_i^k = TI_i^k(1) + IP_i^k(dp) + ES_i^k(dp) : (TI_i^k(1) + dp < k * T_i) \text{ OR } (dp < IP_i^k(dp+1) + ES_i^k(dp))$$

2.- The real starting time will be the initial time instant plus de delay period

Implementation:

```
function rst=RST(k,i,Tasks,ticksiz)

    Tasks=1/ticksiz*Tasks;
    oldticksiz=ticksiz;
    ticksiz=1;
    iti=TI(1,k,i,Tasks);
    ip=0;
    es=0;
    dp=ip+es;

    while
        ceil(((iti+dp))*1/(ticksiz/2))<ceil((k*Tasks(1,i))*1/(ticksiz/2)) | dp<IP(dp+(ticksiz/2),k,i,Tasks)+es

        dp=dp+(ticksiz/2);
        ip=IP(dp,k,i,Tasks);
        es=ES(dp,k,i,Tasks,(ticksiz/2));
        dp=ip+es;

    end

    rst=round(iti+dp);

end
```

4. Conclusions

In this report we have presented a set of procedures to calculate the real starting time of the k -instance of the task i given a set of schedulable *tasks* under RMA. In addition, these procedures have been integrated in the real-time control systems simulator we use in order to be able to evaluate the effects that real time scheduling algorithms have in control systems performance.

References

- Audsley, N.C., A. Burns, R.I. Davis, K.W. Tindell, and J. Wellings (1995). Fixed Priority Pre-emptive Scheduling: An historical Perspective. *Real-Time Systems, the International Journal of Time-Critical Computing Systems*, **Vol. 8**, Number 2/3
- Eker J. and A. Cervin (1999) A Matlab Toolbox for Real-Time and Control Systems Co-Design, *Proc. 6th Int. Conf. on Real-Time Computing Systems and Applications*, Hong Kong, China, December
- Liu, C. and J. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *J.ACM*, **20**, 46-61.
- Ramamritham, K. and J.A. Stankovic (1994). Scheduling Algorithms and Operating Systems Support for Real-Time. *Systems Proceedings of the IEEE*, **Vol. 82**, NO.1, January 1994.
- Stankovic, J.A. (1988). Misconceptions About Real-Time Computing: A Serious Problem for Next Generation Systems. *IEEE Computer*. **21**(10):10-19.
- Tindell K. and H. Hansson (1997). *Real Time Systems by Fixed Priority Scheduling*. Technical report, Department of computer systems, Uppsala University.