Chapter 2

Schedulability Analysis of Tasks in Single Processor Systems: Review of Relevant Work

In this chapter we survey some relevant results for the priority-based schedulability analysis of real-time tasks, both for the fixed and for the dynamic priority assignment schemes. We give emphasis to the worst-case response time analysis in non pre-emptive contexts, since that analysis is of paramount importance to the message schedulability analysis in communication networks.

2.1. Introduction

Real-time computing systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced (Stankovic, 1988). There are various examples of real-time computing systems, such as command and control systems, flight control systems or robotics.

A typical real-time computing system has a real-time program running on the system, which reads inputs from input devices, processes these inputs, and often produces outputs to be sent to output devices. The time between the arrival of an input from a device and the completion of the processing for that input is called the response time for the device (Joseph and Pandya, 1986). The relative deadline for the device can be defined as the maximum interval between the instant of the input arrival and the completion of the processing for that input. Hence, the response time for a device must be smaller or equal to its relative deadline.

Assume that each input device is assigned a task (process) of the application program and that the tasks share a same processor. The problem of determining whether the system will meet its peak processing load, or in other words, whether no input from any device will be lost, becomes one of schedulability analysis of tasks (Burns, 1991).

A round-robin scheduling policy ensures that each task gets a share of the processor. However, such an approach may not be suitable for real-time systems. Assume the following example (Krishna and Shin, 1997):

“Consider a computer controlling an aircraft. Among its tasks are maintaining stability and keeping the cabin temperature within acceptable limits. Suppose the aircraft encounters turbulence that makes it momentarily unstable. The computer is then supposed to adjust the control surfaces to regain stability. If we use round-robin scheduling for this application, the computer may switch context partway through making the control adjustments in order to spend time making sure the cabin temperature

is just right. The result may well be a crash, and the fact that the cabin is being maintained at optimum temperature will be scant consolation to the passengers as the airliner falls out the sky. What we want is to give the stability-maintenance task a very high priority, which ensures that when stability is threatened, all other interfering tasks are elbowed out of the way to allow this all-important task enough computer cycles."

It follows that the consideration of priority levels is crucial to a real-time computing system. If different inputs have different response time requirements, we need to consider different priority levels to schedule the related processing tasks. Consider a real-time system, within which several devices are connected at different priority levels to a single processor computer system. An input being processed, will be pre-empted when another input of higher priority arrives, and will only be resumed when there is no processing remaining at higher priorities.

Assume that the input from a device is saved in a buffer, until it is overwritten by the next input of the same device. The problem is to determine whether for a given assignment of priority levels, the system will meet its peak processing load (i.e. no input from any device will be lost). A more basic problem is how to assign devices to priorities in order to meet the system-processing load.

The remainder of this chapter is organised as follows. In Section 2.2 we outline some of the classic concepts of real-time systems. These aspects include the characterisation of the tasks and the description of the most commonly used priority assignment schemes. As throughout this thesis we will deal with offline schedulability analysis, in Section 2.3 we provide a brief comparison between the main two approaches for performing such schedulability analysis: based on the utilisation of the processor; based on the actual response time of the tasks. In Sections 2.4 and 2.5 we survey the most important results for the schedulability analysis of tasks in single processor systems, for the case of fixed and dynamic priority assignment, respectively. In both cases of priority assignment schemes, we present feasibility tests based on the utilisation of the processor and on the task's response time, and both for pre-emptive and non pre-emptive contexts.

2.2. Classical Concepts of Real-Time Systems

2.2.1. Characterisation of Tasks

In the previous section we mentioned that, in the simplest case, input devices produce inputs at regular intervals. However, in distributed computer-controlled systems (DCCS) not all devices operate in such manner. For example, some may have minimum and maximum time intervals between consecutive inputs, and others may even produce inputs at random intervals. As a consequence, tasks can be characterised according to their predictability. As it will be seen, this characteristic of the tasks affects their schedulability analysis.

Concerning the predictability, three basic types of tasks can be defined: periodic, aperiodic and sporadic.

Periodic tasks, as their name implies, are released on a regular basis. They are characterised by their period, their deadline and their required execution time per period.

The deadline is often assumed to be equal to the period (the processing of an input must be completed, at most, before the next input from the same device).

Aperiodic tasks are released only occasionally, and are usually triggered by an external event. To allow worst-case calculations to be made, a minimum period between any two aperiodic inputs (from the same device) is often defined. If this is the case, the task involved is said to be sporadic, and its period corresponds to its minimum inter-arrival time.

Tasks can also be characterised according to their criticality, depending on the consequences of not being executed before their deadlines. Concerning their criticality real-time tasks can be soft, hard or safety-critical real-time tasks.

Hard real-time tasks are those whose timely execution is critical. If deadlines are missed, severe faults may occur in the system. If the fault is catastrophic, the task is said to be a safety-critical real-time task. Time-utility functions are used in (Burns, 1991) to characterise the types of tasks (Fig. 2.1). For a hard real-time task, if the computation is completed before the deadline, the result will be fully useful; otherwise, it will not have any utility. For a safety-critical real-time task, if the computation is completed before the deadline, the result will be fully useful; otherwise it will have a negative utility.

In most large real-time systems, including DCCS, not all tasks will be hard or safety-critical. Some will even have no deadlines associated, and others will have merely soft deadlines. Soft real-time tasks are, as the name implies, not critical to the application. However, they do deal with time-varying data and hence the utility of result may diminish as the end of computation overpasses the deadline; but remain always positive.

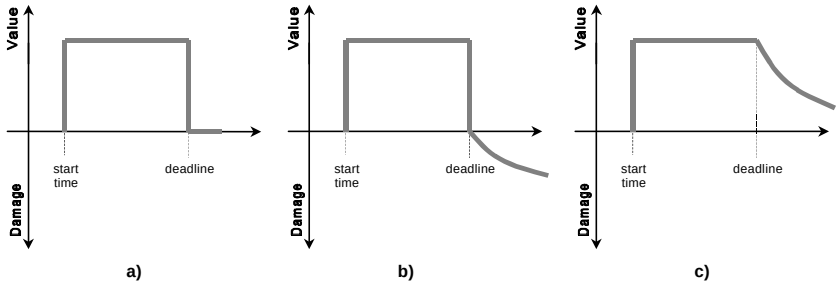

Fig. 2.1 a), b) and c) illustrate the time-utility function for a hard real-time task, a safety-critical real-time task and a soft-real time task, respectively

2.2.2. Scheduling Tasks in Real-Time Systems

Scheduling involves the allocation of time (and resources) to tasks, in such a way that timing requirements (or other performance requirements) are met. Scheduling has been perhaps the most widely research topic within real-time systems. As a consequence, there are multiple taxonomies for the scheduling schemes and for the methodologies for the schedulability analysis.

In a single processor computing system, a set of tasks shares a common resource: the processor. Schedulability analysis has to be performed to predict whether the tasks will meet their timing constraints.

The schedulability analysis can be performed online or offline (Fig. 2.2). In the first case the schedulability of the task set is analysed at run-time, whereas in the latter it is performed prior to run-time (pre-run-time schedulability analysis). In (Ramamritham and Stankovic, 1994) the authors use a different notation: dynamic and static scheduling, to denote systems that perform and do not perform the schedulability analysis at run-time, respectively. The offline scheduling has several advantages over the online scheduling: it requires little run time overhead and the schedulability of the task set is guaranteed before execution. However, it requires a prior knowledge of the tasks' characteristics, which fortunately is possible in most of real-time systems. If the tasks' characteristics are not known prior to run time, schedulability analysis must be performed online. There are basically two types of online schedulers (Ramamritham and Stankovic, 1994): planning-based and best-effort schedulers. In the former, when a new task arrives, the scheduler tries to re-define a new schedule, which is able to comply with both the requirements of the new task and the requirements of the previously scheduled tasks. The new task is only accepted for execution if the schedule is found feasible. In the latter, when a new task arrives, the scheduler does not try to perform a new schedule. The new task is accepted for execution, and the systems tries to do its best to meet deadlines. However, no guarantees are provided for the new coming task, as it may be aborted during execution.

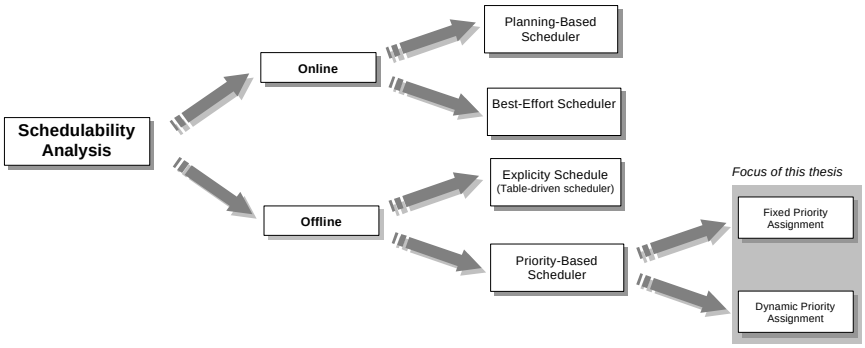

Fig. 2.2 This figure classifies some of the most important types of schedulability analysis

Two types of offline scheduling paradigms are also described in the literature, depending on whether the schedulability analysis produces itself a schedule (or plan) according to which tasks are dispatched at run-time. The table-driven approach (or cyclic executive) is the best known example of an offline scheduling that produces a schedule. The major drawback of the table-driven approach is that it imposes severe restrictions on the period of the tasks (Locke, 1992).

The priority-based approach is one example of offline scheduling where no explicit schedule is constructed. At run-time, tasks are executed in a highest-priority-first basis. Priority-based approaches are much more flexible and accommodating than table-driven approaches.

In the remainder of this chapter we will focus our attention on the offline scheduling paradigms, for tasks dispatched according to priority-based schemes. We assume the following notation (Burns and Wellings, 1996):

Table 2.1: Notation Used for the Schedulability Analysis of Tasks

Notation	***Description***
C	Worst-case computation time of the task
T	Minimum time between task releases (period)
D	Relative deadline of the task
P	Priority level assigned to the task
R	Worst-case response time of a task
U	Utilisation of the task (C/T)
N	Number of tasks in the system

2.2.3. Priority Assignment Schemes

One of the most used priority assignment schemes is to give the tasks a priority level based on its period: the smaller the period, the higher the priority; that is, $T_i < T_j \Rightarrow P_i > P_j$. This assignment is intuitively explained by the fact that more critical devices will provide inputs more frequently (via asynchronous interrupts), or will be polled more frequently. Thus, if they have smaller periods, their worst-case response time must also be smaller. This type of priority assignment is known as the rate monotonic (RM) assignment, and the related pre-run-time schedulability analysis was firstly introduced in (Liu and Layland, 1973).

If some of the tasks are sporadic, it may not be reasonable to consider the relative deadline equal to the period. A different priority assignment can then be to give the tasks a priority level based on its relative deadline: the smaller the relative deadline, the higher the priority; that is, $D_i < D_j \Rightarrow P_i > P_j$. This type of priority assignment is known as the deadline monotonic (DM) assignment (Leung and Whitehead, 1982).

In both RM and DM priority assignments, priorities are fixed, in the sense that they do not vary along time. At run-time, tasks are dispatched highest-priority-first. A similar dispatching policy can be used if the task, which is chosen to run, is the one with the earliest deadline. This also corresponds to a priority-driven scheduling, where the priorities of the tasks vary along time. Thus, the earliest deadline first (EDF) is a dynamic priority assignment scheme. Pre-run-time schedulability analysis for tasks dispatched according to the EDF assignment scheme was also introduced in (Liu and Layland, 1973).

In all three cases, the dispatching phase will take place either when a new task is released or the execution of the running task ends.

2.2.4. Pre-emptive and Non Pre-emptive Systems

In a priority-based scheduler, a higher-priority task may be released during the execution of a lower-priority one. If the tasks are being executed in a pre-emptive context, the higher-priority task will pre-empt the lower-priority one. Contrarily, in a non pre-emptive context, the lower-priority task will be allowed to complete its execution before the higher-priority task starts execution. This situation can be described as a priority inversion due to non pre-emption (a higher-priority task is delayed by a lower-priority one).

2.2.5. Characteristics of the Priority Assignment Schemes

The EDF priority assignment scheme has several advantages over the fixed priority assignment schemes (Spuri, 1996). A first advantage is that it always achieves higher processor utilisation, as was demonstrated in (Liu and Layland, 1973). Additionally, and contrarily to the RM or DM, EDF as been shown to be optimal when arbitrary deadlines are assumed, that is when the relative deadlines are allowed to be greater than the period of the tasks (Lehoczky, 1990).

On the other hand, fixed priority assignment schemes have some important advantages over EDF. Indeed, the EDF dispatching policy is computationally more demanding at run time. Although this aspect has more impact for the task scheduling in single processor environments, it should not be discarded for message scheduling in communication networks (Zuberi and Shin, 1995; Meshi *et al.*, 1996). Most hard real-time systems also have soft real-time components, which can execute at lower priority levels. In EDF, these tasks may occasionally delay execution of more stringent tasks (Sha *et al.*, 1991). Another important drawback of the EDF dispatching policy is its inability to deal with transient overloads (for instance due to exceptions or error recovery actions), since in such a situation some tasks may not meet their deadlines. Contrarily, with a fixed priority assignment approach, a subset of the more critical tasks would still be able to meet their deadlines. With an EDF approach, this is much more difficult to achieve (Buttazzo and Stankovic, 1993). At last, but not least, the analytical methods for computing worst-case response times are much more complex in the case of EDF, even though they are to be used offline.

Consider the following example (Table 2.2), which illustrates the differences between RM and EDF scheduling. We assume relative deadlines equal to the tasks' periods.

Table 2.2: Task Set Example A

Task	*Computation Time (C)*	*Period (T)*
A	35	80
B	10	55
C	5	20

Fig. 2.3 illustrates a time-line (Gantt chart) of the schedule for this task set, assuming that all of them share a common initial release time (at time instant 0), and the tasks are pre-emptable. In Fig 2.3, a) and b) represent the time-lines for a RM-based and an EDF-based schedule, respectively.

2.3. Approaches for the Pre-Run-Time Schedulability Analysis

Real-time computing systems with tasks dispatched according to a priority-based policy (we consider only RM/DM or EDF), must be tested a-priori in order to check if, during run time, no deadline will be lost. This feasibility test is called the pre-run-time schedulability analysis of the task set.

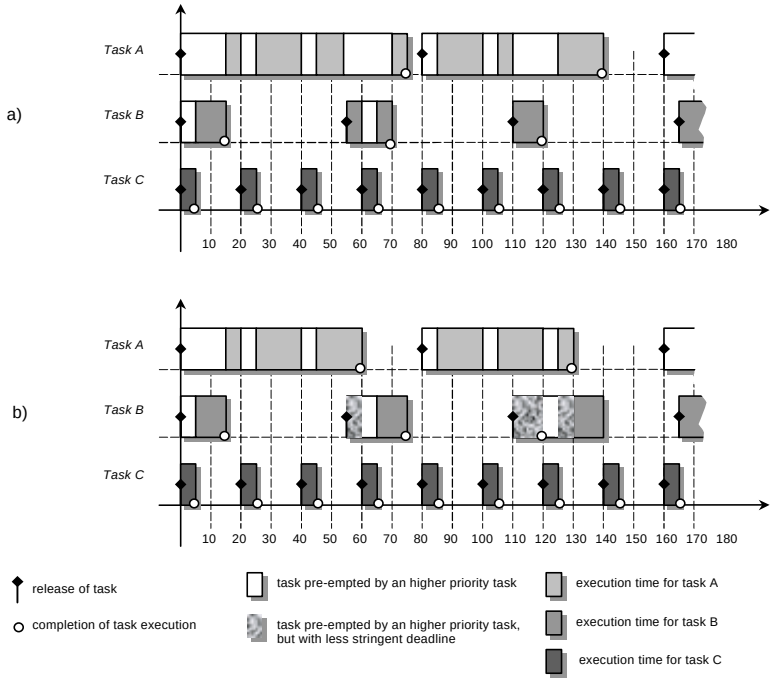

Fig. 2.3 This figure illustrates the schedule for the tasks characterised by Table 2.2, according to a) the RM priority assignment scheme, and b) the EDF priority assignment scheme. Note that in the EDF schedule task *B* is occasionally delayed by task *A*

It can be shown that for periodic tasks, a set of tasks is schedulable if and only if there is a feasible schedule for the LCM (least common multiple) of the periods (Lawler and Martel, 1981). Moreover, it can also be shown that if the tasks share a common request time (known as the critical instant), it is a pre-run-time schedulability sufficient condition that the tasks are schedulable for the longest of the periods (Liu and Layland, 1973). This suggests that a time-line could be used to perform the schedulability analysis. For instance, and concerning the example shown in Table 2.2, where the longest period is 80, Fig. 2.3 shows that the schedule generated by both RM and EDF schemes are feasible for the task set (if all the tasks share a common initial release time). However, time-line approaches may not be effective for systems with a large number of tasks. Hence, analytical methods are preferable.

There are mainly two types of analytical methods to perform pre-run-time schedulability analysis. One is based on the analysis of the processor utilisation. The other is based on the response time analysis for each individual task. In (Liu and Layland, 1973), the authors demonstrated that by considering only the processor utilisation of the task set, a test for the pre-run-time schedulability analysis could be obtained.

Contrarily, a response time test must be performed in two stages. First, an analytical approach is used to predict the worst-case response time of each task. The values obtained are then compared, trivially, with the relative deadlines of the tasks.

The utilisation-based tests have a major advantage: it is a simple computation procedure, which is applied to the overall task set. By this reason, they are very useful for implementing schedulers that check the feasibility online. However, utilisation-based tests have also important drawbacks, when compared with their response-time counterparts. They do not give any indication of the actual response times of the tasks. More importantly, and apart from particular task sets, they constitute sufficient but not necessary conditions. This means that if the task set passes the test, the schedule will meet all deadlines, but if it fails the test, the schedule may or may not fail at run-time (hence, there is a certain level of pessimism). It is also worth mentioning that the utilisation-based tests cannot be used for more complicated task models (Tindell, 1992).

In the next two sections, we survey the most relevant feasibility tests for task sets scheduled both with fixed and dynamic priority schemes, and for both pre-emptive and non pre-emptive contexts. Depending whether the tests are applied to the overall task set or individually to each task, they are classified as utilisation-based tests or response time tests, respectively.

2.4. Feasibility Tests: Case of the Fixed Priority Assignment

2.4.1. Basic Utilisation-Based Test

For the RM priority assignment, Liu and Layland introduced an utilisation-based pre-run-time schedulability test, which, when satisfied, guarantees that tasks will always be completely executed before their deadlines:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \times \left(2^{1/N} - 1\right) \qquad \textbf{(2.1)}$$

This utilisation-based test is valid for periodic independent tasks, with relative deadlines equal to the period, and for pre-emptive systems. As mentioned in the previous section, typically the utilisation-based tests are sufficient but not necessary conditions. For instance, for the task set shown in Table 2.2, the test fails ($0.87 < 0.78$ is false), but the task set is schedulable, as can be seen by the time-line of Fig. 2.3a.

In (Lehoczky *et al.*, 1990), the authors provide an exact analysis, as given below[1]:

$$\min_{(k,l)\in R_i} \left\{ \sum_{j=1}^{i} \left(U_j \times \frac{T_j}{l \times T_k} \times \left\lceil \frac{l \times T_k}{T_j} \right\rceil \right) \right\} \leq 1, \quad \forall_{i,\ 1 \leq i \leq n} \qquad \textbf{(2.2)}$$

where $R_i = \{(k,l)\}$ with $1 \leq k \leq i$ and $l = 1, ..., \lfloor T_i/T_k \rfloor$.

[1] The ceiling function $\lceil x \rceil$ is used to denote the smaller integer greater than or equal to x. Similarly, the floor function $\lfloor x \rfloor$ is used to denote the larger integer smaller than or equal to x.

It is clear that inequality (2.2) is not an easy to use utilisation-based test, hence loosing one of the advantages inherent to the more basic formulations: its simplicity.

2.4.2. Extended Utilisation-Based Tests

Formulations for the utilisation-based tests with deadlines smaller than periods are not available, to our best knowledge. It is however possible to formulate simple utilisation-based test for the case of non pre-emptive tasks.

In (Sha *et al.*, 1990), the authors update the basic utilisation based test (2.1) to include blocking periods, during which higher-priority tasks are blocked by lower-priority ones, to solve the problem of non-independence of tasks (for instance tasks that share resources which are protected by mutual exclusion):

$$\left(\sum_{i=1}^{i}\frac{C_i}{T_i}\right)+\frac{B_i}{T_i}\le i\times\left(2^{1/i}-1\right),\quad \forall_{i,\,1\le i\le N} \tag{2.3}$$

where B_i is the maximum blocking a task i can suffer (Sha *et al.*, 1990). Inequality (2.3) assumes that $P_{i+1} \le P_i$, $\forall_{i<N}$; that is, tasks are ordered by decreasing priority.

In a non pre-emptive context, a higher-priority task can also be "blocked" by a lower-priority task. Assuming that the tasks are completely independent, the maximum blocking time a task can suffer is given by:

$$\begin{cases} B_i = 0, & \text{if } P_i = \min\limits_{j=1,..,N}\{P_j\} \\ B_i = \max\limits_{j\in lp(i)}\{C_j\}, & \text{if } P_i \ne \min\limits_{j=1,..,N}\{P_j\} \end{cases} \tag{2.4}$$

where *lp(i)* denotes the set of lower-priority tasks (than task *i*).

Therefore, inequality (2.3) can be used as an utilisation-based test for a set of non pre-emptable but independent tasks, with the blocking for each task as given by (2.4). Moreover, accepting an increased level of pessimism, inequality (2.4) can be updated to an even simpler formulation:

$$\left(\sum_{i=1}^{N}\frac{C_i}{T_i}\right)+\max_{i,\,1\le i\le N}\left\{\frac{B_i}{T_i}\right\}\le i\times\left(2^{1/N}-1\right) \tag{2.5}$$

Note that if all tasks have the same computation time, (2.5) considers that each task may be blocked at the rate of the highest-priority task.

2.4.3. Response Time Tests for the Pre-emptive Context

In (Joseph and Pandya, 1986) the authors proved that the worst-case response time R_i of a task i is found when all tasks are synchronously released (critical instant) at their maximum rate. R_i is defined as:

$$R_i = I_i + C_i \tag{2.6}$$

where I_i is the maximum interference that task i can experience from higher-priority tasks in any interval $[t, t + R_i)$. The maximum interference (I_i) occurs, when all higher-priority tasks are released synchronously with task i (the critical instant). Without loss of generality, it can be assumed that all processes are released at time instant 0.

Consider a task j with higher-priority than task i. Within the interval $[0, R_i)$, it will be released $\lceil R_i/T_j \rceil$ times. Therefore, each release of task j will impose an interference of C_j. Hence, the overall interference is given by:

$$I_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \quad \textbf{(2.7)}$$

where $hp(i)$ denotes the set of higher-priority tasks (than task i). Substituting this value back in equation (2.2), the worst-case response time R_i of a task t_i is given by:

$$R_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad \textbf{(2.8)}$$

Equation (2.8) embodies a mutual dependence, since R_i appears in both sides of the equation. In fact all the analysis underlay this mutual dependence, since in order to evaluate R_i, I_i must be found, and vice-versa. The easiest way to solve such equation is to form a recurrence relationship (Audsley *et al.*, 1993):

$$W_i^{m+1} = \sum_{j \in hp(i)} \left(\left\lceil \frac{W_i^m}{T_j} \right\rceil \times C_j \right) + C_i \quad \textbf{(2.9)}$$

The recursion ends when $W_i^{m+1} = W_i^m = R_i$, and can be solved by successive iterations starting from $W_i^0 = C_i$. Indeed, it is easy to show that W_i^m is non-decreasing. Consequently, the series either converges or exceeds T_i (in the case of RM) or D_i (in the case of DM). If the series exceeds T_i (or D_i), the task t_i is not schedulable.

2.4.4. Response Time Tests for the non Pre-emptive Context

In (Audsley *et al.*, 1993) the authors updated the analysis of Joseph and Pandya to include blocking factors introduced by periods of non pre-emption, due to the non-independence of the tasks. The worst-case response time is then updated to:

$$R_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad \textbf{(2.10)}$$

which may also be solved using a similar recurrence relationship. B_i is also as given by equation (2.4).

Some care must be taken using equation (2.10) for the evaluation of the worst-case response time of non pre-emptable independent tasks. In the case of pre-emptable tasks, with equation (2.8) we are finding the processor's level-i busy period preceding the completion of task i; that is, the time during which task i and all other tasks with a

priority level higher than the priority level of task *i* still have processing remaining. For the case of non pre-emptive tasks, there is a slight difference, since for the evaluation of the processor's level-*i* busy period we cannot include task *i* itself; that is, we must seek the time instant preceding the execution start time of task *i*.

Therefore, equation (2.6) can be used to evaluate the task's response time of a task set in a non pre-emptable context and independent tasks, where the interference must be now re-defined:

$$I_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{I_i}{T_j} \right\rceil \times C_j \right) \quad \textbf{(2.11)}$$

Consider the following worst-case response time evaluation, assuming the task set shown in Table 2.2, for both pre-emptive and non pre-emptive contexts.

The worst-case response time of task *B* for the pre-emptive context (2.9) is:

$$W_B^0 = 10; \quad W_B^1 = \left(\left\lceil \frac{10}{20} \right\rceil \times 5 \right) + 10 = 15; \quad W_B^2 = \left(\left\lceil \frac{15}{20} \right\rceil \times 5 \right) + 10 = 15$$

Iterations stop at this point since $W_B^2 = W_B^1 = 15$, and thus $R_B = 15$, which coincides with the value given by the time-line (Fig. 2.2).

The worst-case response time of task *B* for the non pre-emptive context (2.11), considering that the blocking is equal to C_A, is:

$$W_B^0 = 35; \quad W_B^1 = 35 + \left(\left\lceil \frac{35}{20} \right\rceil \times 5 \right) = 45; \quad W_B^2 = 35 + \left(\left\lceil \frac{45}{20} \right\rceil \times 5 \right) = 50;$$

$$W_B^3 = 35 + \left(\left\lceil \frac{50}{20} \right\rceil \times 5 \right) = 50$$

Therefore, $R_B = 10 + 50 = 60$. This result shows that the task set example of Table 2.2 is not schedulable in a non pre-emptive context, as task *B* has a response time larger than its period.

Note also that a re-definition for the critical instant must be made. The maximum interference occurs when task *i* and all other higher-priority tasks are synchronously released just after the release of the longest lower-priority task (than task *i*).

2.5. Feasibility Tests: Case of the Dynamic Priority Assignment

2.5.1. Basic Utilisation-Based Test

For the EDF priority assignment, Liu and Layland also introduced an utilisation-based pre-run-time schedulability test (inequality (2.12)).

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \quad \textbf{(2.12)}$$

Similarly to the pre-run-time schedulability test for the RM case (2.1), this result is only valid for sets of non pre-emptive, independent, and periodic tasks, for which the relative-deadline is equal to the period.

Inequality (2.12) can easily be updated to include blocking periods due to the non-independence of the tasks. In (Baker, 1991), the author updated inequality (2.12) to:

$$\left(\sum_{i=1}^{i} \frac{C_i}{T_i} \right) + \frac{B_i}{T_i} \leq 1, \quad \forall_{i,\, 1 \leq i \leq N} \quad \textbf{(2.13)}$$

where B_i is the maximum blocking a task i can suffer, considering the stack resource protocol (SRP). Inequality (2.13) assumes that $T_{i+1} \geq T_i$, $\forall_{i<N}$; that is tasks are ordered by decreasing period.

The key idea behind the SRP is that when a job needs a resource which is not available, it is blocked at the time it attempts to pre-empt, rather than later, when it actually may need the shared resource. This makes inequality (2.13) valid for sets of non pre-emptable tasks, dispatched according to the EDF scheme.

Similarly to the updating of (2.3) to (2.5), inequality (2.13) can be updated to a simpler (but more pessimistic) test:

$$\left(\sum_{i=1}^{N} \frac{C_i}{T_i} \right) + \max_{i,\, 1 \leq i \leq N} \left\{ \frac{B_i}{T_i} \right\} \leq 1 \quad \textbf{(2.14)}$$

where B_i is now defined as:

$$B_i = \max_{j \neq i} \{ C_j \} \quad \textbf{(2.15)}$$

Another relevant result from (Baker, 1991) is that (2.13) can also be extended to task sets within which tasks can have relative deadlines smaller than periods:

$$\left(\sum_{i=1}^{i} \frac{C_i}{D_i} \right) + \frac{B_i}{D_i} \leq 1, \quad \forall_{i,\, 1 \leq i \leq N} \quad \textbf{(2.16)}$$

As a corollary, inequality (2.12) can be extended for task sets within which $D_i \leq T_i$:

$$\sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1 \quad \textbf{(2.17)}$$

These simple utilisation-based tests ((2.14) and (2.16)) are however quite pessimistic. Less pessimistic utilisation-based tests will now be addressed in Sections 2.5.2 and 2.5.3, for pre-emptive and non pre-emptive tasks, respectively. Later, in Sections 2.5.4 and 2.5.5, very recent results on response time analysis will be addressed, for pre-emptive and non pre-emptive tasks, respectively.

2.5.2. Extended Utilisation-Based Tests for the Pre-emptive Context

In (Zheng, 1993) the author extends the results of Liu and Layland in order to consider sporadic tasks, where inequality (2.12) is updated to:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \leq t, \ \forall_{t \geq 0} \qquad \textbf{(2.18)}$$

with $\lceil x \rceil^{+} = 0$ if $x < 0$. The proof for this inequality is intuitive. Assume that at time $t = 0$, there are no pending tasks. Then, a necessary condition to guarantee the tasks' deadlines is that the amount of time, T, needed to transmit all tasks generated during $[0, t]$ with absolute deadlines $\leq t$, is not greater than t. Since the minimum inter-arrival time for a task i is T_i, there are at most $\lceil (t - D_i)/T_i \rceil^{+}$ requests for that task during $[0, t]$ with deadlines $\leq t$. Those requests will need, at most, $\lceil (t - D_i)/T_i \rceil^{+} \times C_i$ time to be completed. Thus, the maximum value for T is given by $\sum_{i=1,..,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i)$. Note that if $D_i = T_i$, inequality (2.18) is satisfied if (2.12) is satisfied, since in this case $\lceil (t - T_i)/T_i \rceil^{+} \leq t/T_i$.

This different formulation has advantages over (2.17), in the sense that it turns out to be a sufficient and a necessary condition (theoretically without any level of pessimism). However, inequality (2.18) can not be classified as a simple test (when compared to (2.17)). It has an additional problem, since it must be checked over an infinite continuous length interval $[0, \infty)$.

However, considering that expression $\sum_{i=1,..,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i)$ does only change at $k \times T_i + D_i$ time instants, inequality (2.18) does only need to be checked for these time instants. Consider the task set example given by Table 2.3.

Table 2.3: Task Set Example B

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*	*Utilisation (U)*
A	30	80	60	0.375
B	10	40	40	0.250
C	5	25	15	0.200

For this task set example, the left-hand side of inequality (2.18) is plotted against its right-hand side (Fig. 2.4), and thus the task set is schedulable by the EDF priority assignment in a pre-emptive context.

Although the consideration of steps for the evaluation of inequality (2.19) eases its use, the problem still remains for the upper limit for t. Different authors have addressed this issue. It is possible to prove that if the total utilisation of the processor is ≤ 1 (condition (2.12)), it exists a point $t_{\max}$, such that $\sum_{i=1,..,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i) \leq t$ always hold for $\forall_{t \geq t_{\max}}$. Consequentely, inequality (2.18) can be re-written as follows:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \leq t, \ \forall_{t \in S}, \ \text{with } S = \left(\bigcup_{i=1}^{N} \{ D_i + k \times T_i, k \in \aleph \} \right) \cap [0, t_{\max}) \qquad \textbf{(2.19)}$$

In (Baruah *et al.*, 1990a) and (Baruah *et al.*, 1990b) the authors demonstrated that t_{max} could be given by $(U/(1\text{-}U)) \times \max_{i=1,...,N}\{(Ti\text{-}Di)\}$, where U represents the overall processor's utilisation ($\sum_{i=1,...,N} (C_i/T_i)$). This result was further improved in (Ripoll *et al.*, 1996), where the upper limit for t is defined as $t_{max} = ((\sum_{i=1,...,N} (1\text{-}D_i/T_i) \times C_i)/(1\text{-}U)$.

Although this last formulation gives a smaller value for t_{max}, it still suffers from the same disadvantage: as the overall utilisation approaches 1, t_{max} becomes very large.

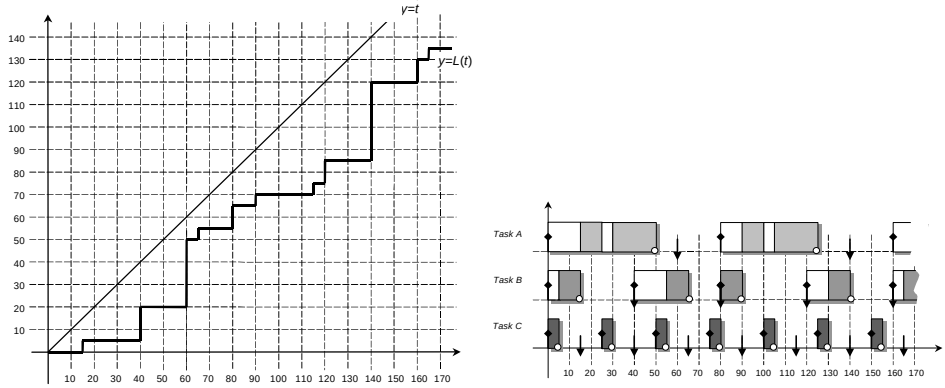

Fig. 2.4 This figure illustrates a time-line (b) for the synchronous *asap* release pattern of task set shown in Table 2.3. In a), the left-hand side of inequality (2.18), denoted as $L(t)$, is represented

For this reason, another approach is considered in (Rippoll *et al.*, 1996) and (Spuri, 1995), where the authors demonstrate that $t_{max} = L$ (synchronous processor's busy period). The synchronous processor's busy period is defined as the time interval from the critical instant up to the first instant when there are no more pending tasks in the system. For instance, for the time-line shown in Fig. 2.4b), $L = 65$. Analytically, L can be found as follows:

$$L = \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil \times C_i \qquad \textbf{(2.20)}$$

Equation (2.20) is solved by recurrence, starting with $L^0 = \sum_{i=1,..,N} C_i$. When $L^{m+1} = L^m = L$, the solution has been found (note that this recurrence relationship converges if, and only if condition (2.12) is verified). For the task set of Table 2.3, it follows that:

$$L^0 = 45; \quad L^1 = \left\lceil \frac{45}{80} \right\rceil \times 30 + \left\lceil \frac{45}{40} \right\rceil \times 10 + \left\lceil \frac{45}{25} \right\rceil \times 5 = 30 + 20 + 10 = 60;$$

$$L^2 = \left\lceil \frac{60}{80} \right\rceil \times 30 + \left\lceil \frac{60}{40} \right\rceil \times 10 + \left\lceil \frac{60}{25} \right\rceil \times 5 = 30 + 20 + 15 = 65;$$

$$L^3 = \left\lceil \frac{65}{80} \right\rceil \times 30 + \left\lceil \frac{65}{40} \right\rceil \times 10 + \left\lceil \frac{65}{25} \right\rceil \times 5 = 30 + 20 + 15 = 65$$

and iterations stop, as $L^3 = L^2 = L = 65$.

2.5.3. Extended Utilisation-Based Tests for the non Pre-emptive Context

For the non pre-emptive context, a similar test was presented in (Zheng, 1993) and (Zheng and Shin, 1994):

$$\sum_{i=1}^{N}\left\lceil\frac{t-D_i}{T_i}\right\rceil^{+}\times C_i+\max_{j=1,\ldots,N}\{C_j\}\le t,\ \forall_{t\ge D_{\min}},\ \text{ with }\ D_{\min}=\min_{j=1,\ldots,N}\{D_j\} \qquad \textbf{(2.21)}$$

Comparing to the test for the pre-emptive context (2.18), the inclusion of the blocking factor is intuitive (see Section 2.5.1.). However, in (George *et al.*, 1996) the authors discuss the pessimism inherent to the inequality (2.21). The main argument is that in this inequality it is considered that the cost of possible priority inversions is always initiated by the longest task and, moreover, it is effective during the entire interval under analysis. To reduce this level of pessimism, they suggest the following modification:

$$\sum_{i=1}^{N}\left\lceil\frac{t-D_i}{T_i}\right\rceil^{+}\times C_i+\max_{\substack{j=1,\ldots,N\\ D_j>t}}\{C_j\}\le t,\ \ \forall_{t\in S},\quad \text{with}\ \max_{\substack{j=1,\ldots,N\\ D_j>t}}\{C_j\}=0\ \text{ if }\nexists_j : D_j>t \qquad \textbf{(2.22)}$$

That is, the blocking factor is only included if its deadline occurs after *t*.

Considering that the execution time of a task is expressed as a multiple of the system's tick, the blocking task must start its execution one tick before the critical instant. As a consequence, in the diverse formulations which have been including blocking factors due to the system's non pre-emptability, such blocking could be expressed as (C_i - 1).

2.5.4. Response Time Tests for the Pre-emptive Context

The worst-case response time analysis for pre-emptive EDF scheduling was first introduced in (Spuri, 1996). The starting point for such analysis was that the worst-case response time for a general task set is not necessarily obtained considering the critical instant, as defined for the fixed priority case. In his work, Spuri demonstrated that the worst-case response time of a task *i* is found in the processor's deadline-*i* busy period (analogous to the processor's level-*i* busy period in the case of fixed priorities). However, the longest processor's deadline-*i* busy period may occur when all tasks but task *i* (contrarily to the case of fixed priority assignment) are synchronously released and at their maximum rate.

This means that, in order to find the worst-case response time of task *i*, we need to examine multiple scenarios within which, while task *i* has an instance released at time *a*, all other tasks are synchronously released at time *t* = 0. As an example, consider the task set shown in Table 2.5.

Table 2.5: Task Set Example C

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*	*Utilisation (U)*
A	1	4	4	0.250
B	2	6	6	0.333
C	2	9	9	0.222
D	2	15	15	0.133

Considering that all tasks are synchronously released at time instant 0, then a time-line is as shown in Fig. 2.5.

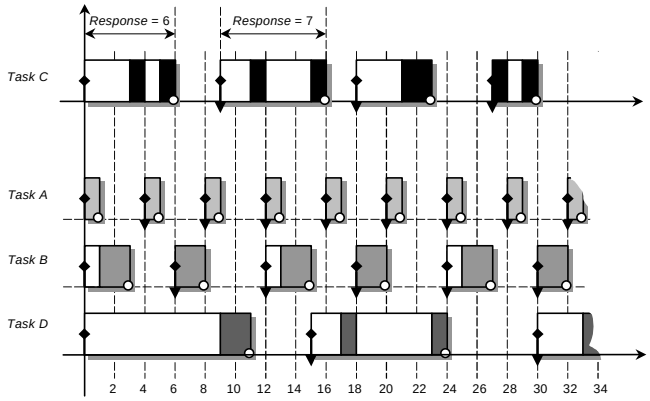

Fig. 2.5 This time-line illustrates the fact that, when evaluating the synchronous busy period, the worst-case response time does not occur for the first instance of task *C*

From Fig. 2.5 we can conclude that the instance of task *C* which is released at time instant $t = 9$ ($a = 9$) has a higher response time than the instance which is released at $t = 0$ ($a = 0$). Thus, given a value of a, the response of an instance of task i, which is released at time a, is:

$$R_i(a) = \max\{C_i, L_i(a) - a\} \tag{2.23}$$

where $L_i(a)$ is the length of the deadline-i busy period, which starts at time instant $t = 0$. $L_i(a)$ can be evaluated by the following iterative computation:

$$L_i(a) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \left(\min\left\{ \left\lceil \frac{L_i(a)}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} \times C_j \right) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor \right) \times C_i \tag{2.24}$$

Equation (2.17) can be solved by recurrence, starting with $L_i^0(a) = 0$. When $L_i^{m+1}(a) = L_i^m(a) = L_i(a)$, the solution has been found. Obviously, in equation (2.24), the computational load only considers tasks that have deadlines earlier than D_i. Consider the task set example of Table 2.6.

Table 2.6: Task Set Example D

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*
A	1	4	4
B	2	10	10

Consider that $a = 0$. At time instant $t = 9$, and for task *B*, the number of instances released for task *A* is 3 ($\lceil 9/4 \rceil = 3$). However, from those 3 releases, only the first 2 have absolute deadlines earlier than the deadline of task *B* (since $1 + \lfloor (10 - 4)/4 \rfloor = 2$). Assume again the scenario of Table 2.5. Using equation (2.24), with $a = 0$, the evaluation of the response time for task *C* is:

$$L_C^0(0)=0; \quad L_C^1(0)=0+(1+0)\times 2=2;$$
$$L_C^2(0)=(\min\{1,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=5;$$
$$L_C^3(0)=(\min\{2,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=6;$$
$$L_C^4(0)=(\min\{2,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=6$$

Substituting this result back into equation (2.23), gives $R_C(0)$ = 6. The same computation, but now a = 9 (also illustrated in the time-line given by Fig. 2.5) gives:

$$L_C^0(0)=0; \quad L_C^1(0)=0+(1+1)\times 2=4;$$
$$L_C^2(0)=(\min\{1,4\})\times 1+(\min\{1,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=9;$$
$$L_C^3(0)=(\min\{3,4\})\times 1+(\min\{2,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=13;$$
$$L_C^4(0)=(\min\{4,4\})\times 1+(\min\{3,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=16;$$
$$L_C^5(0)=(\min\{4,4\})\times 1+(\min\{3,3\})\times 2+(\min\{2,1\})\times 2+(1+1)\times 2=16$$

Substituting this result back into equation (2.23), gives $R_C(9)$=max {2, (16 - 9)} = 7, and thus it is now clear that the worst-case response time of a task i is not necessarily found when all tasks are synchronously released.

Finally, in the general case, the worst-case response time for a given task i is:

$$R_i=\max_{a\geq 0}\{R_i(a)\} \tag{2.25}$$

The remaining problem is how to determine the values of a. Looking to the right-hand side of equation (2.24), we can easily understand that its value only changes at $k \times T_j + D_j - D_i$ steps.

$$a\in\bigcup_{j=1}^{N}\{k\times T_j+D_j-D_i, k\in\aleph_0\}\cap[0,L[\tag{2.26}$$

with L as given by equation (2.20).

2.5.5. Response Time Tests for the non Pre-emptive Context

The worst-case response time analysis for the non pre-emptive EDF scheduling was introduced in (George *et al.*, 1996). The main difference from the analysis for the pre-emptive case is that a task instance with a later absolute deadline can possibly cause a priority inversion. Thus, and similarly to what was said for the fixed priority case (Section 2.4.4), instead of analysing the deadline-i busy period preceding the completion time of task i, we must analyse the busy period preceding the execution start time of the task's instance. Consequently, the response time of the t_i 's instance released at time a is:

$$R_i(a)=\max\{C_i, L_i(a)+C_i-a\} \tag{2.27}$$

where $L_i(a)$ is now the length of the busy period (preceding execution).

Thus, $R_i(a)$ can be evaluated by means of the following iterative computation:

$$L_i(a) = \max_{D_j > a + D_i} \{C_j\} + \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \left(\min \left\{ 1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} \times C_j \right) + \left\lfloor \frac{a}{T_i} \right\rfloor \times C_i \qquad \textbf{(2.28)}$$

This equation may be solved also by recurrence. Note again that the blocking factor could be written as $(C_i - 1)$. Note also that in order to analyse the busy period, the start of execution time, $1 + \lfloor L_i(a) / T_j \rfloor$, is used instead of $\lceil L_i(a) / T_j \rceil$.

2.6. Summary

In this chapter we provide a comprehensive survey of the most relevant results for the pre-run-time schedulability analysis of task sets in single processor systems.

The feasibility tests have been classified as utilisation-based tests and response time tests, according to the information which is provided by its evaluation; that is, in the former and indication is provided on the overall processor utilisation, while on the latter, the actual response time of each individual task is provided as a result.

Feasibility tests for fixed and dynamic priorities (both for the pre-emptive and non pre-emptive contexts) are provided (when available).

The emphasis is given to feasibility tests for non pre-emptable, independent task sets, within which tasks may have deadlines smaller than periods, as they will be the foundation of Chapter 7, where they will be adapted to encompass the characteristics of P-NET and PROFIBUS networks.

Finally, it is important to mention that this chapter is not an extended survey of all the important scheduling aspects, which could be pertinent to DCCS. The presented results are those strictly necessary as the background for the remaining chapters of this thesis. Just to mention some of the aspects which were not addressed in detail in this chapter, we can refer the problem of shared resources (Sha *et al.*, 1990; Rajkumar *et al.*, 1988), the problem of co-operative scheduling (Tindell, 1992; Tindell, 1994; Tindell and Clark, 1994; Palencia and Harbour, 1998), the problem of arbitrary deadlines (Lehoczky, 1990; Tindell and Clark, 1994; Palencia and Harbour, 1998) or the problem of finding the worst-case execution time of tasks (Puschner and Koza, 1989).

2.7. References

Audsley, N., Burns, A., Richardson, M., Tindell, K and Wellings, A. (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. In *Software Engineering Journal*, Vol. 8, No. 5, pp. 285-292.

Baker, T. (1991). Stack-Based Scheduling of Real-Time Processes. In *Real-Time Systems*, Vol. 3, No. 1, pp. 67-99.

Baruah, S., Howell, R., Rosier, L. (1990a). Algorithms and Complexity Concerning the Pre-emptive Scheduling of Periodic Real-time Tasks on One Processor. In *Real-Time Systems*, 2, pp. 301-324.

Baruah, S., Mok, A. and Rosier, L. (1990b). Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS'90)*, pp. 182-190.

Burns, A. (1991). Scheduling Hard Real-Time Systems. In *Software Engineering Journal*, Special Issue on Real-Time Systems, May 1991, pp. 116-128.

Burns, A. and Wellings, A. (1996). Real-Time Systems and Programming Languages. Addison-Wesley, 2nd Edition.

Buttazzo, G. and Stankovic, J. (1993). RED: Robust Earliest Deadline Scheduling. In *Proceedings of the 3rd International Workshop on Responsive Computing Systems*.

George, L., Rivierre, N., Spuri, M. (1996). Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling. Technical Report No. 2966, INRIA.

Joseph, M. and Pandya, P. (1986). Finding Response Times in a Real-Time System. In *The Computer Journal*, Vol. 29, No. 5, pp. 390-395.

Krishna, C. and Shin, K. (1997). Real-Time Systems. McGraw-Hill Series in Computer Sciences.

Lawler, E. and Martel, C. (1981). Scheduling Periodically Occurring Tasks on Multiple Processors. In *Information Processing Letters*, Vol. 12, No. 1, pp. 9-12.

Lehoczky, J., Sha, L. Ding, Y. (1989). The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166-171**.**

Lehoczky, J. (1990). Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 201-209**.**

Leung, J. and Whitehead, J. (1982). On the Complexity of fixed-priority Scheduling of Periodic Real-Time Tasks. In *Performance Evaluation*, Vol. 22, No. 4, pp. 237-250.

Liu, C. and Layland, J. (1973). Scheduling Algorithms for Multiprograming in Hard-Real-Time Environment. In *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61.

Locke, C. (1992). Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. In *Real-Time Systems*, Kluwer Academic Publishers, Vol. 4, No. 1, pp. 37-53.

Meshi, A., Natale, M. and Spuri, A. (1996). Earliest Deadline Message Scheduling with Limited Priority Inversion. In *Proceedings of the Workshop on Parallel and Distributed Real Time Systems*.

Puschner, P. and Koza, C. (1989). Calculating the Maximum Execution Time of Real-Time Programs. In *Real-Time Systems*, Vol. 1, No. 2, pp. 159-176.

Palencia, J. and Harbour, M. (1998). Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of IEEE Real-Time Systems Symposium*, pp. 26-37.

Rajkumar, R., Sha, L., and J. Lehoczky (1988). Real-Time Synchronisation Protocols for Multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 259-269.

Ramamritham, K. and Stankovic, J. (1994). Scheduling Algorithms and Operating Systems Support for Real-Time Systems. In *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 55-67.

Ripoll, I., Crespo, A., Mok, A. (1996). Improvement in Feasibility Testing for Real-time Systems. In *Real-Time Systems*, 11, pp. 19-39.

Sha, L., Rajkumar, R. and J. Lehoczky (1990). Priority Inheritance Protocols: an Approach to Real-Time Synchronisation. In *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1175-1185.

Sha, L., Klein, M. and Goodenough, J. (1991). Rate Monotonic Analysis for Real-Time Systems. Carnegie Mellon University, Technical Report CMU/SEI-91-TR-6 1991.

Spuri, M. (1995). Earliest Deadline Scheduling in Real-time Systems. PhD Thesis, Scuola Superiore Santa Anna, Pisa.

Spuri, M. (1996). Analysis of Deadline Scheduled Real-Time Systems. Technical Report No. 2772, INRIA.

Stankovic, J. (1988). Real-Time Computing Systems: the Next Generation. In *Tuturial: Hard Real-Time Systems*, Stankovic, J. and K. Ramamritham (Editors), IEEE Computer Society Press, Los Alamitos, USA, pp. 14-38.

Tindell, K. (1992). An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Department of Computer Science, University of York, Technical Report YCS-189.

Tindell, K. (1994). Adding Time-Offsets to Schedulability Analysis. Department of Computer Science, University of York, Technical Report YCS-221.

Tindell, K. and Clark, J. (1994). Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. In *Microprocessors and Microprogramming*, Vol. 40, pp. 117-134.

Zheng, Q. (1993). Real-Time Fault-Tolerant Communication in Computer Networks. PhD Thesis, University of Michigan.

Zheng, Q., Shin, K. (1994). On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. In *IEEE Transactions on Communications*, Vol. 42, No. 2/3/4, pp. 1096-1105.

Zuberi, K. and Shin, K. (1995). Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In *Proceedings of Real-Time Technology and Applications Symposium*, pp. 240-249.