4

RESPONSE TIMES UNDER EDF SCHEDULING

In the previous chapter it has been shown how to analyze a set of independent real-time tasks, in order to assess their feasibility when scheduled by the EDF algorithm. In almost all of its formulations, this feasibility assessment problem has been solved by examining the schedule of a subset of task instances in a bounded interval of time. It has been shown that if any deadline is missed in this interval, then any other instantiation of the same task set is not feasibly scheduled by EDF.

A different solution to feasibility analysis is found by introducing the notion of *worst case response time*: the worst case response time of a task is the maximum time elapsed between the release and the completion times of any of its instances. An alternative solution to the feasibility assessment problem would be to compute, for each task, the worst case response time and to compare it with its relative deadline.

However, it is worth remarking that the two problems, feasibility assessment and worst case response time computation, are not equivalent when EDF scheduling is assumed. In fixed priority systems, the feasibility analysis of a task set is indeed carried out by computing task worst case response times [12]. In contrast, when EDF scheduling is assumed the two problems have a different complexity. The feasibility problem is simpler, since it can be solved by examining a single specific set of instances. Instead, the worst case response time computation requires the analysis of several different scenarios, although it too has a pseudo-polynomial time complexity.

In spite of being slightly more complex, the problem of determining the task worst case response times is very interesting. In particular, it is a very useful

tool not only for the analysis of uni-processor systems, but also for the analysis of distributed real-time systems (see Chapter 9). Distributed applications are, in fact, characterized by precedence relationships among their tasks. If the tasks are statically allocated to single processors, *end-to-end* timing constraints can be analyzed by a theory which assumes release jitter [1]: "All tasks are defined to arrive at the same time, but a precedence constrained task on one processor can have its release delayed awaiting the arrival of a message from its predecessors. The worst case release jitter of such a subtask can be computed by knowing the response times of predecessor tasks located on other processors." This approach leads to a global analysis of distributed real-time systems, which is termed *holistic schedulability analysis* by Tindell and Clark [13]. It is clear that the central issue of such a distributed analysis is the ability to compute task worst case response times in the presence of release jitter.

In this chapter an algorithm for the computation of task worst case response times is described, assuming the processor is scheduled according to the EDF algorithm. The algorithm can also be used to extend the holistic approach to the analysis of deadline scheduled distributed systems [10] which is done in Chapter 9. Extensions to the algorithm for handling more complex, but still independent task models on a uni-processor, and an extensive case study are discussed at the end of this chapter.

4.1 FINDING LOCAL MAXIMA

The worst case response time rt_i of a task τ_i is the maximum time between a τ_i's instance arrival and its completion. Finding rt_i is not a trivial task when EDF scheduling is assumed. Contrary to what may be an intuitive idea, the worst case response time of a task is not always found in the initial busy period of a synchronous schedule, which is thus not exactly the equivalent of the *critical instant* under fixed priority scheduling [6]. However, the concept of busy period is still useful.

The idea is that the completion time of a task's instance with deadline d must be the end of a busy period in which all executed instances have deadlines less than or equal to d. An argument similar to that of Theorem 3.14 can then be applied in order to locally maximize the task response time, for a given subset of all possible scenarios.

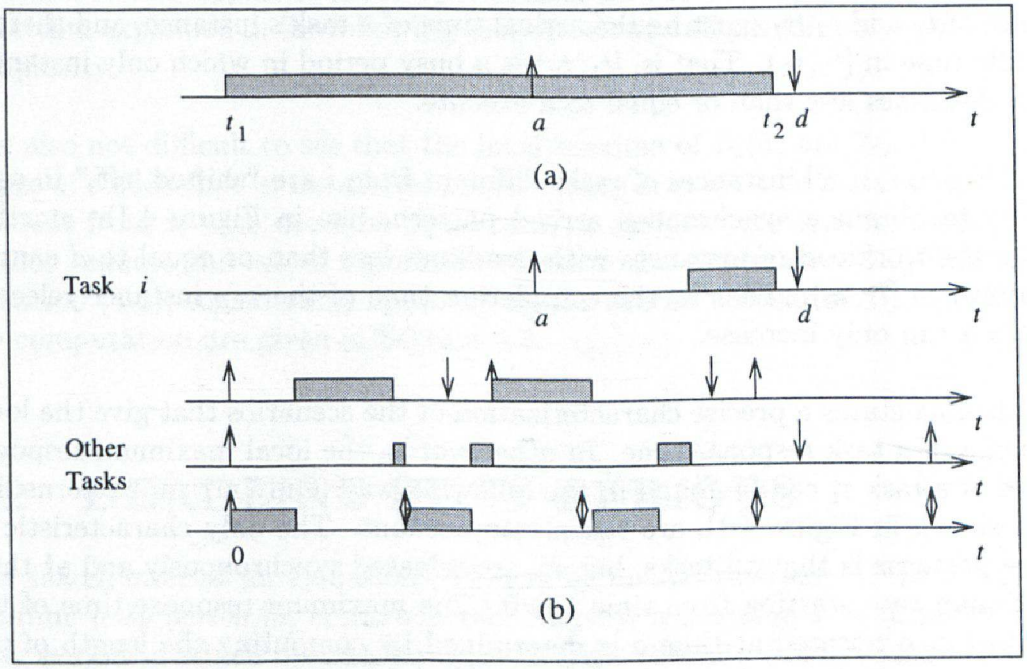

Figure 4.1a Busy period preceding an instance completion time.

Figure 4.1b Synchronous arrival pattern possibly giving the worst case response time for task τ_i.

Lemma 4.1 (Spuri) *The worst case response time of a task τ_i is found in a busy period in which all other tasks are released synchronously at the beginning of the period and then at their maximum rate (see Figure 4.1b).*

Proof. Consider a τ_i's instance, like in Figure 4.1a, with arrival time a and deadline $d = a + D_i$, respectively. Let t_2 be its completion time, according to the EDF schedule. Let t_1 be the last time before t_2, such that there are no pending instances with arrival time earlier than t_1 and deadline less than or equal to d. Since τ_i's instance is released at $t = a$, $t_1 \leq a$. Furthermore, by choice of t_1 and t_2, t_1 must be the arrival time of a task's instance, and there is no idle time in $[t_1, t_2)$. That is, $[t_1, t_2)$ is a busy period in which only instances with deadlines less than or equal to d execute.

At this point, if all instances of tasks different from i are "shifted left," in such a way to obtain a synchronous arrival pattern, like in Figure 4.1b, starting at t_1, the workload of instances with deadlines less than or equal to d cannot diminish in $[t_1, t_2)$. That is, the completion time of the τ_i's instance released at $t = a$ can only increase. □

The lemma states a precise characterization of the scenarios that give the local maxima of a task response time. In other words, the local maximum response times of a task τ_i can be found in the following way. Only arrival patterns like that shown in Figure 4.1b are taken into account. The only characteristic of these patterns is that all tasks, but τ_i, are released synchronously and at their maximum rate starting from time $t = 0$. The maximum response time of the τ_i's instance released at time a is determined by computing the length of the busy period of all task instances with deadlines less than or equal to d, that is, of all task instances that precede the τ_i's instance being considered.

The parameter that distinguishes these patterns is the release time a. Accordingly, let $L_i(a)$ be the length of the busy period relative to the deadline $d = a + D_i$, and $rt_i(a)$ be the maximum response time relative to a. Since C_i is an obvious lower bound for any τ_i's instance response time, it is

$$rt_i(a) = \max\{C_i, L_i(a) - a\}. \tag{4.1}$$

The worst case response time of τ_i is finally

$$rt_i = \max_{a \geq 0} \{rt_i(a)\}. \tag{4.2}$$

Put in this way, Equation (4.2) is not quite useful. In order to make it practical for actual tools, an upper bound on the significant values of the parameter a

is needed. Furthermore, the upper bound has to be small enough to make the complexity of the procedure at least pseudo-polynomial in time.

The argument of Lemma 4.1 indirectly answers this question, too. Accordingly, the maximum response time of a task is found in a busy period, and it is known that the longest busy period of a given task set is the initial one in a completely synchronous arrival pattern (refer to Section 3.2.5). Hence, the significant values of a are in the interval $[0, L - C_i)$. In Section 3.2.5 it is shown that when the processor utilization of the task set is bounded by a fixed positive constant less than 1, L is upper bounded by a pseudo-polynomial value. This gives the algorithm explained in the following sections an overall pseudo-polynomial time complexity.

It is also not difficult to see that the local maxima of $L_i(a)$ are found for those values of a such that in the arrival pattern there is at least an instance of a task different from τ_i with deadline equal to d, or all tasks are synchronized. This further reduces the set of significant values to be examined when evaluating Equation (4.2), considerably speeding up the overall procedure. The details of the computation are given in Section 4.3.

4.2 DEADLINE BUSY PERIODS

The search interval for Equation (4.2) can be further restricted if the notion of *deadline busy period* [5] is introduced. Namely, a deadline d busy period is a busy period in which only task instances with absolute deadline smaller than or equal to d execute.

By looking at the argument given in the proof of Lemma 4.1, it can be easily realized that the local maxima of a task response times are actually found in deadline busy periods. That is, $L_i(a)$ denotes the length of the deadline $a + D_i$ busy period found in the EDF schedule of the arrival pattern in which all tasks but τ_i are released synchronously, while τ_i has its first instance released at time

$$s_i(a) = a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i$$

(so that to have an instance released at time $t = a$).

The computation of $L_i(a)$ can be carried out by means of a useful recursive approach. In particular, given the described arrival pattern, by time t,

$\lceil t/T_j \rceil$ instances of τ_j are released, for each $j \neq i$. However, at most only $1 + \lfloor (a + D_i - D_j)/T_j \rfloor$ of them can have a deadline less than or equal[1] to $d = a + D_i$. That is, the "higher priority workload" having arrived by time t is

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min \left\{ \left\lceil \frac{t}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j,$$

and the number of τ_i's instances released up to time $t = a$ is

$$1 + \left\lfloor \frac{a}{T_i} \right\rfloor.$$

The length $L_i(a)$ can then be computed with the following iterative formula:

$$\begin{cases} L_i^{(0)}(a) &= 0, \\ L_i^{(m+1)}(a) &= W_i\left(a, L_i^{(m)}(a)\right) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i. \end{cases} \tag{4.3}$$

As for Equation (3.1), the convergence of Equation (4.3) in a finite number of steps is ensured by the condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

The reason is that the maximum length of any busy period is L, the length of the initial busy period of the synchronous arrival pattern. At each step of Equation (4.3), either $L_i^{(m+1)}(a)$ is equal to $L_i^{(m)}$, in which case the computation is halted, or $L_i^{(m+1)}(a)$ is $L_i^{(m)}$ increased at least by a quantity $C_{\min}$, the minimum computation time among all tasks. Since $L_i(a)$ is bounded by L, its value is thus reached in a finite number of steps.

As remarked by George, et. al. [5], not all deadline busy periods are interesting, but only those that include the reference instance. Namely, given a τ_i's instance released at time $t = a$, hence with deadline $d = a + D_i$, the deadline d busy period includes this instance if and only if it is longer than a.[2] The length of the longest such deadline busy periods is denoted by L_i. L_i is clearly an upper bound for the significant values of the variable a in Equation (4.2). Its computation can be sped up by utilizing the following two properties.

[1] Note that no particular policy is assumed for breaking deadline ties. Hence, in the worst case, instances sharing the same deadline should be considered as having higher priorities.

[2] Note that when $L_i(a) \leq a$ the arrival pattern is not interesting with respect to the worst case response time computation.

Maximum Deadline Busy Period Lengths(τ):
$L_{n+1} = L$;
for $i = n$ **downto** 1
let k be such that $e_k \leq L_{i+1} - C_i + D_i < e_{k+1}$;
a $= e_k - D_i$;
while $L_i(a) \leq a$
let k be such that $e_k \leq L_i(a) - C_i + D_i < e_{k+1}$;
a $= e_k - D_i$;
endwhile
$L_i = L_i(a)$;
endfor
return$(L_1, \ldots, L_n)$

Figure 4.2 Pseudo-code of the algorithm for the computation of the maximum deadline busy period lengths.

Lemma 4.2 (George, et. al.)

$$D_i \leq D_j \quad \Rightarrow \quad L_i \leq L_j.$$

Lemma 4.3 (George, et. al.) $\forall i$, $L_i(a)$ *is non-decreasing in* a.

Let $E = \bigcup_{i=1}^{n} \{mT_i + D_i : m \geq 0\} = \{e_1, e_2, \ldots\}$, let L be the length of the initial busy period of the synchronous arrival pattern, and assume that $\forall i$, $D_i \leq D_{i+1}$. The algorithm for the computation of L_i starts by placing an instance of τ_i in such a way to have its deadline at the largest e_k smaller than or equal to $L_{i+1} - C_i + D_i$ (as previously remarked, these are the patterns that give the longest deadline busy periods). If the resulting deadline e_k busy period includes this last instance of τ_i, L_i is found, otherwise the computation is repeated by choosing a smaller deadline, this time according to the value of $L_i(a)$. Note that the algorithm always stops. The pseudo-code is reported in Figure 4.2.

4.3 ALGORITHM DESCRIPTION

Once the bound L_i has been computed for any task τ_i, the worst case response times can be computed by evaluating the following equation:

$$rt_i = \max_{a \in A} \{rt_i(a)\}.$$

A is the set of the significant values for the parameter a. As previously addressed, it is the set of instants for which $a + D_i$ coincides with at least the absolute deadline of another task's instance. Namely,

$$A = \left(\bigcup_{j=1}^{n} \{kT_j + D_j - D_i : k \geq 0\} \right) \bigcap [0, L_i) .$$

Finally, $rt_i(a)$ is computed by means of Equation (4.1) and the iterative formula (4.3).

Note that as for the problem of deciding the feasibility of a task set, the complexity of the worst case response time computation is pseudo-polynomial whenever the processor utilization of the task set is upper bounded by a fixed positive constant smaller than 1 [9]. Similarly, whether the problem has a fully polynomial solution is an open question.

4.4 EXTENDED TASK MODELING

In Section 3.2.7, the feasibility analysis for hybrid task sets was extended to more complex models in which release jitter, sporadically periodic tasks, tick scheduling, and non-preemptive scheduling are, in turn, taken into account. These extensions, as well as a new one termed *deadline tolerance*, are now discussed with respect to worst case response times computation.

4.4.1 Deadline Tolerance

The concept of *deadline tolerance* is introduced by Buttazzo and Stankovic [2, 3]. The idea is to accept a schedule as feasible even if some deadline is missed. However, the possible lateness of a task instance must not exceed a certain value called tolerance.

Definition 4.1 *A task τ_i has* deadline tolerance δ_i *if any of its instances with deadline d must complete within $d + \delta_i$.*

Note that for a task instance, having a tolerance greater than zero is different from having a longer deadline. The deadline is the point in time by which the

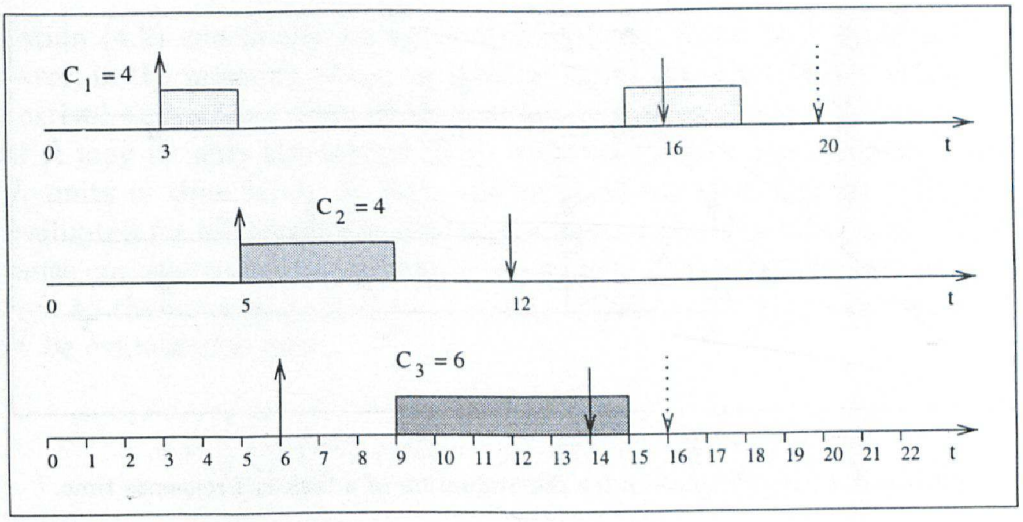

Figure 4.3 EDF schedule with deadline tolerance.

instance should complete, and that can be used by the operating system to drive the scheduling algorithm, while the tolerance is the maximum lateness tolerated in case of a late job.

In Figure 4.3 an example of EDF schedule with deadline tolerance is shown. The first and the last jobs both miss their respective deadlines. However, since they have enough tolerance, 4 and 2, respectively, the schedule is accepted as feasible.

If each task is allowed to have its own tolerance, the analysis of the system becomes very difficult. In fact, the amount of lateness that a job can have depends strictly on the sequence of job arrivals. A possible solution is a run-time solution as proposed in [2], where at each job arrival the complete schedule of the current job set is examined in order to evaluate each possible lateness. This algorithm is useful when the jobs executions can be dynamically rejected, but it isn't if the feasibility of the task set has to be checked *a priori*.

In the latter case, an obvious solution is now given by the worst case response time computation. In order to check the feasibility of a task τ_i, it suffices to compute its maximum lateness $l_i = \max\{0, rt_i - D_i\}$, and to compare it with the task tolerance δ_i.

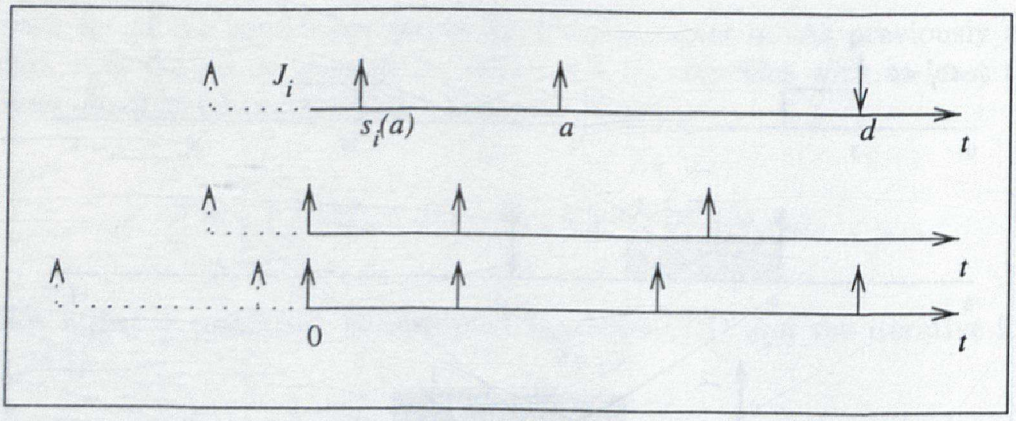

Figure 4.4 Arrival pattern for the evaluation of a task τ_i's response time.

4.4.2 Release Jitter

When the release jitter is introduced in the task model, the computation of worst case response times is simply extended as shown for the feasibility analysis. The argument of Lemma 4.1 can be applied to the extended model too. The difference is again in the most constraining arrival patterns, owing to the initial release jitter. An example of the patterns examined to find the worst case response time of a task τ_i is shown in Figure 4.4.

As previously, given a, all possible τ_i's instances are included in the pattern, so that there is one arrival at time $t = a$. By forcing the first instance to experience a release jitter J_i, the number of τ_i's instances released up to time a now is

$$1 + \left\lfloor \frac{a + J_i}{T_i} \right\rfloor .$$

Similarly, the computation of the higher priority workload arrived up to time t is

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min \left\{ \left\lceil \frac{t + J_j}{T_j} \right\rceil , 1 + \left\lfloor \frac{a + D_i + J_j - D_j}{T_j} \right\rfloor \right\} C_j .$$

The length of the resulting busy period relative to the deadline $d = a + D_i$ can still be computed by means of the iterative computation (4.3), accordingly modified. The worst case response time relative to a is then

$$rt_i(a) = \max\{J_i + C_i, L_i(a) - a\}.$$

Equation (4.2) can finally be applied to find rt_i. There is a slight difference, however, in the meaning of the variable a: in the previous case a denoted both the arrival and release time of the instance considered; in presence of release jitter it may be only the arrival time, with the release time possibly being up to J_i units of time later. In fact, the value of $rt_i(a)$ in Equation (4.2) must be evaluated for all significant values of a such that the release time of the τ_i's instance considered is greater than or equal to 0. Thus the computation can be limited to the interval $[-J_i, L_i - J_i - C_i)$. Namely, the set A on which $rt_i(a)$ must be evaluated is now

$$A = \left(\bigcup_{j=1}^{n} \{kT_j + D_j - J_j - D_i : k \geq 0\} \right) \bigcap [0, L_i) .$$

4.4.3 Sporadically Periodic Tasks

The extension to sporadically periodic tasks can be handled with a very similar approach. Assuming the definitions of $I_i(t)$ and $H_i(t)$ given in Section 3.7, the number of τ_i's instances released up to time a is $H_i(a+D_i)$. Also, the number of τ_j's instances which have deadlines before or at $a+D_i$ is $H_j(a+D_i)$. Similarly, the number of τ_j's instances released by time t is $I_j(t)$. The higher priority workload having arrived by time t is thus

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min \{I_j(t), H_j(a + D_i)\} C_j .$$

The rest of the procedure, *i.e.*, the computation of the busy period relative to $d = a + D_i$, the computation of the worst case response time relative to a, and the computation of the overall worst case response time, remains basically unchanged. Refer to [9] for a full description.

4.4.4 Tick Scheduling

Applying the same argument as for Theorem 3.15, also Lemma 4.1 can be generalized according to the mixed scheduling model, in which the cost of an actual scheduler implementation is taken into account. That is, the worst case response times can still be evaluated with the described approach. However, the new availability function must be considered. In practice, only the definition of $W_i(a,t)$, the higher priority workload arriving by time t, needs to be modified

by taking into account the additional term due to the tick scheduler overheads:

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min \left\{ \left\lceil \frac{t + J_j}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i + J_j - D_j}{T_j} \right\rfloor \right\} C_j + OV(t).$$

All the rest is unchanged. As for the previous section, refer to [9] for a more detailed description.

4.4.5 Non-Preemptive Non-Idling EDF Scheduling

In Section 3.8 it has been shown that the feasibility check of a task set scheduled by a non-preemptive non-idling EDF scheduler is very similar to the preemptive case. As expected, the similarity is also valid for the computation of task worst case response times, as reported in [5]. There are, however, two differences worth mentioning:

- The first one is also addressed in Theorem 3.16. Owing to the absence of preemption, a task instance with a later absolute deadline can cause a priority inversion, which must be accounted for.
- The second difference is in some way more subtle. Always owing to the non-preemptability of any task instance execution, the attention has to be on the busy period preceding the execution start time of the instance, and not on the busy period preceding its completion time, as is the case in the preemptive model. The reason is that when preemption is not allowed, once a task instance has gained the processor, it cannot be preempted, even if a higher priority instance becomes ready during its execution.

Before showing the details of the worst case response times computation, it is first necessary to characterize the scenarios, and in particular the relative deadline busy periods, which provide the local response times maxima.

Lemma 4.4 (George, et. al.) *The worst case response time of a task τ_i is found in a deadline busy period for τ_i in which τ_i has an instance released at time a (and possibly others released before), all tasks with relative deadline smaller than or equal to $a+D_i$ are released from time $t = 0$ on at their maximum rate, and finally a further task with relative deadline greater than $a+D_i$, if any, has an instance released at time $t = -1$.*

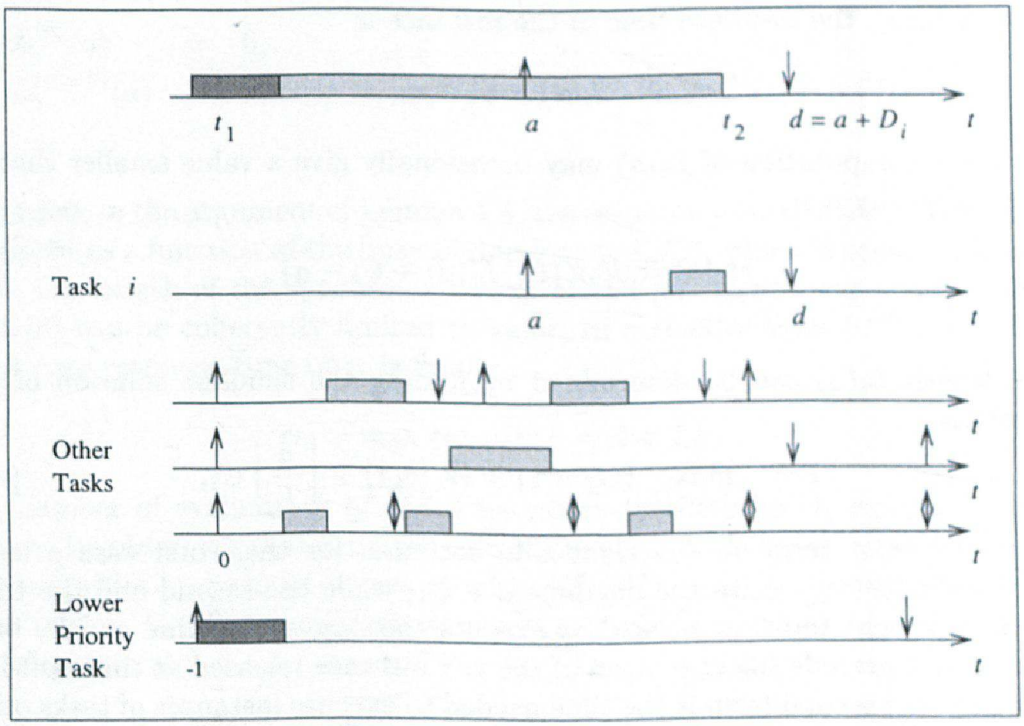

Figure 4.5 Local response time maxima when non-preemptive EDF scheduling is assumed.

In Figure 4.5 an example of such scenario is depicted. Note that, with respect to the deadline $a + D_i$, there can be at most a single priority inversion before the corresponding deadline busy period. Other inversions may be present in this busy period. However, since it includes by definition only instances with deadlines less than or equal to $a + D_i$, possible inversions between these instances are irrelevant as far as the τ_i's instance released at time a is concerned. All have to be executed anyway before it, regardless of their execution schedule.

As suggested by the lemma, if the length of the busy period starting at time $t = 0$ and preceding the execution of the τ_i's instance released at time a is termed $L_i(a)$, the response time of the instance is

$$L_i(a) + C_i - a.$$

Since the computation of $L_i(a)$ may occasionally give a value smaller than a, more generally it is

$$rt_i(a) = \max\{C_i, L_i(a) + C_i - a\}.$$

The length $L_i(a)$ can be determined by finding the smallest solution of the equation

$$t = \max_{D_j > a + D_i} \{C_j - 1\} + \bar{W}_i(a, t) + \left\lfloor \frac{a}{T_i} \right\rfloor C_i, \tag{4.4}$$

where the first term on the right side accounts for the worst case priority inversion with respect to the deadline $a + D_i$, while the second and the third terms represent the time needed to execute the largest deadline $a + D_i$ busy period that precede the execution of the τ_i's instance released at time a. More precisely, the second term is the time needed to execute instances of tasks other than τ_i with absolute deadlines smaller than or equal to $a + D_i$ and release times before this τ_i's instance execution start time. Finally, the third term is the time needed to execute the τ_i's instances released before a.

The rationale of the equation is to compute the time needed by the τ_i's instance released at time a to get the processor: every other higher priority instance released before this event is executed earlier, thus its execution time must be accounted for. For the same reason, the function $\bar{W}_i(a, t)$ must account for all higher priority instances released in the interval $[0, t]$, thus also including those possibly released at time t. For any task τ_j, the maximum number of instances released in $[0, t]$ is $1 + \lfloor t/T_j \rfloor$.

However, at most $1 + \lfloor (a + D_i - D_j)/T_j \rfloor$ among them can have an absolute deadline before or at $a + D_i$. It follows that

$$\bar{W}_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min \left\{ 1 + \left\lfloor \frac{t}{T_j} \right\rfloor , 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j .$$

Since $\bar{W}_i(a,t)$ is a monotonic non-decreasing step function, the smallest solution of Equation 4.4 can be found by using the following fixed point computation:

$$\begin{cases} L_i^{(0)}(a) & = & 0, \\ L_i^{(m+1)}(a) & = & \max_{D_j > a + D_i} \{C_j - 1\} + \bar{W}_i \left(a, L_i^{(m)}(a) \right) + \left(\left\lfloor \frac{a}{T_i} \right\rfloor \right) C_i . \end{cases}$$

According to the argument of Lemma 4.4, the response time relative to a, $rt_i(a)$, is defined as a function of the busy period length $L_i(a)$, which is upper bounded by L, the length of the synchronous busy period [5]. Hence, the computation of $rt_i(a)$ can be coherently limited to values of a smaller than L. That is, the worst case response time of τ_i is finally

$$rt_i = \max \{ rt_i(a) : 0 \leq a < L \} .$$

The number of evaluations of $rt_i(a)$ necessary to compute rt_i can be further reduced by observing that the right side of Equation 4.4 is a step function whose discontinuities in a are for values equal to $kT_j + D_j - D_i$, for some task τ_j and some integer k. The significant values of a in the interval $[0, L)$ are reduced accordingly.

Moreover, as for the preemptive case, it is possible to further restrict the interval where the worst case response time of τ_i has to be looked for, to $[0, L_i]$, with L_i being the maximum length of a deadline busy period including a τ_i's instance (see [5] for a detailed description).

Note also that, similarly to the feasibility condition, the computation of worst case response times has pseudo-polynomial time complexity, since L_i is upper bounded by L, whose length is pseudo-polynomial whenever $U \leq c$, with c a positive constant smaller than 1 [5].

Semaphore	Locked by	Time held
1	Task 9	900
2	Task 9	300
2	Task 15	1350
3	Task 6	400
3	Task 10	400
4	Task 3	100
4	Task 9	300
5	Task 11	750
5	Task 15	750

Table 4.1 List of semaphores and locking pattern for the GAP task set.

4.5 CASE STUDY

The theory presented in this Chapter has been applied to the GAP (Generic Avionics Platform) task set, a small avionics case study described by Locke *et. al.* in [7]. There are seventeen tasks in the GAP set, of which all but one are strictly periodic, with periods multiple of T_{tick}, therefore they do not suffer release jitter. Task 11 is a sporadic task, whose arrival is assumed to be polled by the tick scheduler, hence it may suffer a worst case release jitter equal to T_{tick}. Some tasks also share resources that are accessed by locking and unlocking semaphores according to a hypothetical pattern, which is assumed to be equal to that described by Tindell *et. al.* in [12], and which is reported in Table 4.1. Similarly, a tick scheduler is assumed with the same parameters as in their description, that is,:

$$C_{tick} = 66\mu s \quad T_{tick} = 1000\mu s \quad C_{QL} = 74\mu s \quad C_{QS} = 40\mu s.$$

According to Tindell *et. al.*'s approach [12], for this task set there is no optimal priority assignment able to guarantee all tasks under a fixed priority system. However, according to the EDF analysis, the GAP task set is indeed feasible under EDF scheduling. The worst case response times computed with the theory described in the paper are reported in Table 4.2, where all times are given in microseconds. Note that all tasks with the same relative deadline have the same worst case response time. This case study demonstrates an example of the potential value to EDF scheduling over fixed priority scheduling.

i	D_i	T_i	C_i	B_i	J_i	rt_i	a_i
1	5000	200000	3000	0	0	4180	0
2	25000	25000	2000	300	0	12280	0
3	25000	25000	5000	300	0	12280	0
4	40000	40000	1000	300	0	20226	40000
5	50000	50000	3000	400	0	30226	30000
6	50000	50000	5000	400	0	30226	30000
7	59000	59000	8000	400	0	39226	21000
8	80000	80000	9000	1350	0	60226	0
9	80000	80000	2000	1350	0	60226	0
10	100000	100000	5000	1350	0	74150	0
11	200000	200000	1000	1350	1000	168558	0
12	200000	200000	3000	0	0	168558	0
13	200000	200000	1000	0	0	168558	0
14	200000	200000	1000	0	0	168558	0
15	200000	200000	3000	0	0	168558	0
16	1000000	1000000	1000	0	0	198760	0
17	1000000	1000000	1000	0	0	198760	0

Table 4.2 GAP task set parameters.

4.6 SUMMARY

This Chapter presented solutions for computing the worst case execution times for tasks. These worst case response times can then be compared to task deadlines to check for feasibility. Another advantage of this approach is that it is extensible to distributed real-time computing as shown in Chapter 9.

REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, September 1993.

[2] G.C. Buttazzo and J.A. Stankovic, "RED: A Robust Earliest Deadline Scheduling Algorithm," *Proc. of 3rd Int. Workshop on Responsive Computing Systems*, 1993.

[3] G.C. Buttazzo and J.A. Stankovic, "Adding Robustness in Dynamic Preemptive Scheduling," in *Responsive Computer Systems: Toward Integration of Fault Tolerance and Real-Time*, Kluwer Press, 1994.

[4] L. George, P. Muhlethaler, and N. Rivierre, "Optimality and Non-Preemptive Real-Time Scheduling Revisited," Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France, 1995.

[5] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling," Rapport de Recherche RR-2966, INRIA, Le Chesnay Cedex, France, 1996.

[6] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the Association for Computing Machinery* 20(1), 1973.

[7] Locke C.D., Vogel D.R., and Mesler T.J., "Building a Predictable Avionics Platform in Ada: A Case Study," *Proc. of IEEE Real-Time Systems Symposium*, 1991.

[8] M. Spuri, "Earliest Deadline Scheduling in Real-Time Systems," Doctorate Dissertation, Scuola Superiore S.Anna, Pisa, Italy, 1995.

[9] M. Spuri, "Analysis of Deadline Scheduled Real-Time Systems," Rapport de Recherche RR-2772, INRIA, Le Chesnay Cedex, France, 1996.

[10] M. Spuri, "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems," Rapport de Recherche RR-2873, INRIA, Le Chesnay Cedex, France, 1996.

[11] J.A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, June 1995.

[12] K. Tindell, A. Burns, and A.J. Wellings, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks," *Real-Time Systems* 6(2), 1994.

[13] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," *Microprocessors and Microprogramming* 40, 1994.