On Optimal, Non-Preemptive Scheduling of Periodic Tasks

Kevin Jeffay*
University of Washington
Department of Computer Science FR-35
Seattle, WA  98195

Technical Report 88-10-03
October 1988

* Supported in part by a Graduate Fellowship from the IBM Corporation, and in part by a grant from the National Science Foundation (number CCR-8700435).
On Optimal, Non-Preemptive Scheduling of Periodic Tasks

Kevin Jeffay*
University of Washington
Department of Computer Science FR-35
Seattle, WA 98195

October 1988

1. Overview

This note examines the problem of non-preemptively scheduling a set of real-time tasks on a uniprocessor. In section two we present our real-time task model. Next we present a non-preemptive Earliest Deadline First (EDF) scheduling algorithm and prove necessary and sufficient conditions for the EDF algorithm to always schedule a set of real-time tasks correctly. We also demonstrate that these conditions are necessary for the correctness of any non-preemptive algorithm which never unnecessarily idles the processor and therefore establish the optimality of the non-preemptive EDF algorithm.

2. Real-Time Tasking Model

A real-time system is composed of a set of tasks $\tau = \{T_i = (s_i, c_i, p_i) \mid \forall i, 1 \leq i \leq n; c_i \leq p_i\}$, where

$s_i =$ Start time: the time of the first request for execution of task $i$.

$c_i =$ Computational cost: the time to execute task $i$ to completion on a dedicated uniprocessor. We assume that this execution time is a constant and that it is the same for all execution requests of $T_i$.

$p_i =$ Period: the interval between requests for execution of task $i$. The period is also assumed to be a constant.

We assume all tasks in our system are periodic. If a task is periodic with period $p$ and the task makes its initial request for execution at time $s$, then the task will make its $k^{th}$ request for execution exactly at time $s + (k-1)p$. Once activated, tasks make periodic requests for execution forever. We assume all tasks are independent in the sense that their execution

* Supported in part by a Graduate Fellowship from the IBM Corporation, and in part by a grant from the National Science Foundation (number CCR-8700435).
time requests are dependent only upon the time of their last request and not upon those of any other task. We further assume that the starting times \( s_i \) of all tasks are unknown.

Throughout this paper we consider a discrete time model. In this domain we assume that all the \( s_i, c_i \) and \( p_i \) are expressed as integer multiples of some basic indivisible time unit. For convenience we also assume that our set of tasks is sorted in non-decreasing order by period \( p_i \geq p_j \) if \( i > j \). The index of a task refers to its position in this sorted list.

Before presenting the non-preemptive EDF algorithm we first define what it means for an algorithm to be correct. Given a set of tasks \( \tau \) and a scheduling algorithm \( A \), we say \( A \) is correct with respect to \( \tau \) if for all possible assignments to the \( s_i \), every task \( T_i \) is guaranteed to have completed its \( k^{th} \) request for execution before the \((k+1)^{st}\) request is made. We define a request interval for task \( T_i \) to be an interval in real-time \([s_i+(k-1)p_i, s_i+kp_i]\), for some request number \( k \).

3. The Non-Preemptive EDF Algorithm

The basic algorithm we consider is the non-preemptive Earliest Deadline First (EDF) algorithm. Intuitively, a deadline for a task is a point in time before which the task must complete its execution. For the periodic tasks in our model, each execution request will have a distinct deadline \( D_{ik} = s_i+kp_i \) for the \( k^{th} \) request for execution of task \( T_i \). Prior to its first request for execution at time \( s_i \), \( T_i \) has no deadline \( (D_{i0} = \infty) \). Similarly, a task has no deadline during interval between the time a task has completed execution and the time that task makes its next request for execution. We say task \( T_i \) misses a deadline if there is some request interval \( k \) for which \( T_i \) has not completed execution by time \( s_i+kp_i \).

A high-level pseudo code version of the EDF scheduling algorithm is given below. Although the initial starting time of each task is unknown, assume that there is some clairvoyant process that assigns the correct initial starting times of each task in the initialization loop below. This caveat has no bearing on the operation of the basic algorithm and allows for a clearer presentation. Assume that the variable Time represents the true value of real world time and that it is maintained by a separate, non-resource consuming process.

At every scheduling point the EDF algorithm chooses the task whose deadline is closest to the current point in time\(^1\). Once chosen for execution, a task is executed to completion without preemption.

```plaintext
{ 
  \( R_i \) = time of Task i's next request
  \( D_i \) = time for Task i's current deadline
  \( Time \) = current value of the real-time clock
}

FOR i=1 TO n DO { Initialization }
  \( R_i \leftarrow s_i \)
  \( D_i \leftarrow s_i + p_i \)
END FOR
```

\(^1\)Throughout this paper the expression "task \( i \) is scheduled" means that task \( i \) commences execution. Alternatively we could say task \( i \) is dispatched, initiated etc.
DO forever
{ Main scheduling loop }

\[ j \leftarrow \text{index of task for whom } D_j = \min_{s \in S_n} \min_{k \leq s} D_k \]
{ Assume } j = 0 \{ \text{if no such task exists.}
Ties are broken by picking the task with smallest index. \}

IF \( j = 0 \) THEN
\[ \text{delay until Time} = \min_{s \in S_n} \min_{i \leq s} R_i \]
ELSE
Execute(\( T_j \))
\[ R_j \leftarrow R_j + P_j \]
\[ D_j \leftarrow D_j + P_j \] { Logically, at this point Time has been "increased" by \( c_j \) }
END IF
END DO

With the exception of the delay statement in the main loop above, we assume that this algorithm takes "no time" to execute in our discrete time system.

4. Analysis

We first note that our version of the non-preemptive EDF algorithm always schedules a task if there is a task ready to execute at a scheduling point. A scheduling discipline which intentionally idles the processor when there are outstanding requests for the processor is said to use inserted idle time [Conway et al. 67]. In this section we derive conditions which ensure the correctness of the non-preemptive EDF algorithm on a uniprocessor, and show the optimality of the non-preemptive EDF algorithm with respect to the class of non-preemptive scheduling algorithms which do not use inserted idle time.

While we restrict ourselves to comparison against algorithms without inserted idle time mainly for technical reasons in the proofs that follows, we are confident that this result is useful and important. For example, in the important case where a task set fully utilizes the processor, once all tasks become activated, it is obvious that if an algorithm ever idles the processor it can never be correct.

The following theorem establishes necessary conditions for ensuring the correctness of any non-preemptive discipline which does not use inserted idle time.

**Theorem 1:** Let \( \tau \) be a set of real-time processes \( \{ T_1, T_2, ..., T_n \} \), sorted in non-decreasing order by period with unknown starting times. Let \( A \) be any non-preemptive, scheduling discipline that correctly schedules \( \tau \) without inserted idle time. Algorithm \( A \) can be correct with respect to \( \tau \) only if:

\[ 1) \sum_{i=1}^{n} \frac{c_i}{P_i} \leq 1, \]
2) \( \forall k, 1 \leq k < n: p_k \geq \max \left( c_i \ + \ \max_{0 < l \leq p_i - p_k} \left( -l + \sum_{j=1}^{i-1} \left\lfloor \frac{p_{k+l} - 1}{p_j} \right\rfloor c_j \right) \right) \).

Informally, condition (1) can be thought of as a requirement that the system not be overloaded. If a real-time task \( T \) has a cost \( c \) and period \( p \), then \( \frac{c}{p} \) is the fraction of processor time consumed by \( T \) over the lifetime of the system (i.e., the utilization of the processor by \( T \)). The first condition simply stipulates that the total processor utilization cannot exceed 1.0. The right hand side of the inequality in the second condition is an accounting of the worst case blockage that can occur between the time a task makes a request for execution and the time it is scheduled. If we think of a task's period as its maximum tolerable latency for each request it makes, then condition (2) simply requires that each task have a latency greater than or equal to the worst case blockage that it can experience. The derivation of the worst case blockage is in the proofs below.

**Proof:** (By contradiction.)

We first show that condition (1) is necessary. Assume there exists a set of processes \( \tau \) with

\[
\sum_{i=1}^{n} \frac{c_i}{p_i} > 1,
\]

which is scheduled correctly by algorithm \( A \) for any assignment of values to the \( s_i \).

For all \( i \), let \( s_i = 0 \) and let \( t = \text{LCM}(p_i) \). Let \( u_{a,b} \) be the total processor time consumed by \( \tau \) in the interval \([a,b]\) when scheduled by \( A \). Consider the interval in time \([0,t]\). If algorithm \( A \) schedules these tasks correctly then it must be the case that

\[
u_{0,t} \leq t.
\]

Therefore since \( \frac{t}{p_i} c_i \) is the total processor time spent on task \( i \) in \([0,t]\), we must have

\[
u_{0,t} = \sum_{i=1}^{n} \frac{t}{p_i} c_i \leq t.
\]

However, this implies that

\[
\sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1,
\]

\(^1\) Least Common Multiple.
which is a contradiction of our original assumption.

For condition (2), again assume $\tau$ is scheduled correctly by algorithm $A$ for all possible values of $s_i$, but yet there exists a task $T_k (k < n)$ such that

$$p_k < \text{MAX}_{i > k} \left( c_i + \text{MAX}_{0 < i < p_i, p_k} \left( -l + \sum_{j=1}^{i-1} \left\lfloor \frac{p_k+l-1}{p_j} \right\rfloor c_j \right) \right).$$

Let $i'$ and $l'$ be values of $i$ and $l$ respectfully that maximize the right hand side of the above inequality. Let $s_{i'} = 0$, and $s_j = 1$ for $1 \leq j \leq n, j \neq i', k$. Let $s_k = l' - rp_k$, where $r$ is the largest possible integer such that $l' - rp_k > 0$. This gives rise the pattern of task execution requests shown below. (The shaded areas indicate the intervals in which each execution request would be scheduled by the non-preemptive EDF algorithm.)

If $A$ correctly schedules $\tau$, then it must be the case that in the interval $[0, l' + p_k]$, the total processor time consumed is

$$u_{o, l' + p_k} \geq c_{i'} + \sum_{j=1}^{i'-1} \left\lfloor \frac{p_k+l'-1}{p_j} \right\rfloor c_j.$$

Now by our initial assumption,
\[ c_i' + \sum_{j=1}^{i'-1} \left[ \frac{p_k+l'-1}{p_j} \right] c_j > p_k + l'. \]

and hence

\[ u_{o,l'} p_k > p_k + l'. \]

However, this is impossible since the amount of processor time consumed in any given interval cannot be larger than the length of the interval. Hence algorithm A could not have possibly have correctly scheduled \( \tau \) with our chosen starting times. Therefore we again have a contradiction of our original assumption.

We now demonstrate the optimality of the non-preemptive EDF discipline over all non-preemptive disciplines that do not use inserted idle time. By optimal we mean that if any non-preemptive algorithm that does not use inserted idle time can correctly schedule a set of real-time tasks, then the non-preemptive EDF algorithm will also correctly schedule the tasks. To prove optimality, we show that conditions (1) and (2) are sufficient for ensuring the correctness of the non-preemptive EDF algorithm.

**Theorem 2:** Let \( \tau \) be a set of real-time processes \( \{T_1, T_2, \ldots, T_n\} \), sorted in non-decreasing order by period with unknown starting times. The non-preemptive EDF discipline is correct with respect to \( \tau \) if conditions (1) and (2) from the previous theorem hold.

**Proof:** (By contradiction.)

Assume the contrary, i.e., that conditions (1) and (2) from theorem 1 hold and yet there exists a set of values for the \( s_i \) such that a task \( T_k \) that misses a deadline at some point in time when \( \tau \) is scheduled by the non-preemptive EDF algorithm. Without loss of generality, assume that \( T_k \) is the first task to miss a deadline. (In the case of simultaneous missed deadlines, let \( T_k \) be the task with smallest index.) Consider the first execution request of \( T_k \) that misses a deadline. Let \( t_d \) be the deadline of this request. For the remaining tasks whose start times are less than \( t_d \), either they have a deadline at \( t_d \) or they have a execution request interval that contains \( t_d \).
For the tasks with request intervals that contain \( t_d \), we consider the following two cases:
either none of the request intervals that contain the point \( t_d \) are scheduled prior to \( t_d \), or,
some of the request intervals are scheduled prior to \( t_d \).

**Case 1:** None of the request intervals that contain the point \( t_d \) are scheduled prior to \( t_d \).

Let \( t_0 \) be the end of the last period in which the processor was idle. If the processor has never been idle let \( t_0 = 0 \).

We begin by deriving a bound on the total processor demand in the interval \([t_0, t_d]\). Let \( d_{a,b} \) be the processor demand required by \( \tau \) in the interval \([a, b]\). In the interval \([t_0, t_d]\), the total processor time demand is

\[
d_{t_0, t_d} \leq \sum_{j=1}^{n} \left\lfloor \frac{t_d - t_0}{p_j} \right\rfloor c_j .
\]

This bound is obtained as follows. Since each task scheduled in \([t_0, t_d]\) either does or does not have request intervals that contain one of the endpoints of the interval, we need only consider the four patterns of execution requests shown below.

Since \( t_0 \) was the end of the last idle period, any request interval that contains the point \( t_0 \) must have been scheduled and completed execution prior to \( t_0 \). (If \( t_0 = 0 \) then there are no such overlapping request intervals.) Also, by the premise of this case, all request intervals that contain the point \( t_d \) are scheduled after \( t_d \). Therefore, each task \( T \) scheduled in \([t_0, t_d]\) requires the processor for an amount of time less than or equal to

\[
\left\lfloor \frac{t_d - t_0}{p} \right\rfloor c .
\]

Hence, in the interval \([t_0, t_d]\), the total processor time required by \( \tau \) is
\[ d_{t_0,t_d} \leq \sum_{j=1}^{n} \left\lfloor \frac{t_d-t_0}{p_j} \right\rfloor c_j. \]

Now since there is no idle period in \([t_0,t_d]\) and since a task misses a deadline at \(t_d\), it must be the case that

\[ t_d - t_0 < d_{t_0,t_d}. \]

Hence we have

\[ t_d - t_0 < \sum_{j=1}^{n} \left\lfloor \frac{t_d-t_0}{p_j} \right\rfloor c_j \leq \sum_{j=1}^{n} \frac{t_d-t_0}{p_j} c_j, \]

or simply

\[ t_d - t_0 < \sum_{j=1}^{n} \frac{t_d-t_0}{p_j} c_j \]
\[ < (t_d - t_0) \sum_{j=1}^{n} \frac{c_j}{p_j}. \]

However this implies that

\[ 1 < \sum_{j=1}^{n} \frac{c_j}{p_j}, \]

which is a contradiction of condition (1).

**Case 2:** Some of the request intervals that contain the point \(t_d\) are scheduled prior to \(t_d\).

Let \(T_i\) be a task who does not have a deadline at \(t_d\), and whose request interval that contains \(t_d\) is scheduled prior to \(t_d\). Let \(t_s = t_d - p_k\) (the point in time at which \(T_k\) makes its request for execution that has deadline at \(t_d\)). Let \(t_{s'}\) be the point in time at which \(T_i\) makes its request for execution that contains \(t_d\).

**Claim 1:** \(T_i\) has a greater index than \(T_k\) \((i > k)\).

**Proof of claim:** Assume that this is not the case, that is, \(i < k\) and \(T_i\) is a task with a request interval containing \(t_d\) that is scheduled prior to \(t_d\). If \(i < k\), then \(p_i \leq p_k\) and hence \(t_s < t_{s'} < t_d\) as shown below.
From the definition of \( T_i \), we know \( T_i \) is scheduled during the interval \([t_s; t_d]\). However, note that \( T_k \) has a nearer deadline than \( T_i \) throughout the interval \([t_s'; t_d]\). Therefore, by our construction of the non-preemptive EDF discipline, \( T_k \) must have been scheduled, and hence completed execution, before \( T_i \) had been scheduled. Since \( T_i \) is scheduled prior to \( t_d \), \( T_k \) must have completed execution before its deadline \( t_d \). This a contradiction since \( T_k \) missed its deadline at \( t_d \).

Of all the tasks with request intervals that contain \( t_d \) which were scheduled prior to \( t_d \), let \( T_i \) be the task whose request interval that contains \( t_d \) was scheduled last. Let \( t_i \) be the point in time in which the request interval of \( T_i \) that contains \( t_d \) is scheduled.

**Claim 2:** \( t_i < t_s' \). (The request interval of \( T_i \) that contains \( t_d \) is scheduled before \( t_s' \).)

**Proof of claim:** Again assume that this is not the case and that the request interval of \( T_i \) that contains \( t_d \) is scheduled after \( t_s' \).

In this case, we know \( T_i \) is scheduled during the interval \([t_s'; t_d]\). Again we note that \( T_k \) has a nearer deadline than \( T_i \) throughout the interval \([t_s; t_d]\). Therefore, by our construction of the non-preemptive EDF discipline, \( T_k \) must have been scheduled, and hence completed execution, before \( T_i \) had been scheduled. Since \( T_i \) is scheduled prior to \( t_d \), \( T_k \) must have completed execution before its deadline \( t_d \). This a contradiction since \( T_k \) missed its deadline at \( t_d \).
Proof of claim: Since $T_i$ is the last task with a request interval containing $t_d$ scheduled prior to $t_d$, other than $T_i$, every task scheduled in $[t_i, t_d]$ has a deadline at or before $t_d$. Therefore, if a task $T_j$ that is scheduled in $[t_i, t_d]$ had made a request for execution at $t_i$, the non-preemptive EDF discipline would have scheduled $T_j$ instead of $T_i$ at time $t_i$. Since $T_i$ was scheduled at $t_i$, we can conclude that no task with an outstanding request for execution whose deadline is less than or equal to $t_d$ could have made a request for execution at $t_i$.

Claim 4: No task with index greater than $i$ is scheduled in the interval between the time the request interval of $T_i$ that contains $t_d$ is scheduled and $t_d$ (the interval $[t_i, t_d]$).

Proof of claim: Again, since $T_i$ is the last task with a request interval containing $t_d$ scheduled prior to $t_d$, other than $T_i$, every task scheduled in $[t_i, t_d]$ has a deadline at or before $t_d$. Let $T_j$, where $j > i$, be a task with a request interval that has a deadline somewhere in the interval $[t_i, t_d]$. Since $t_d - t_i < p_i \leq p_j$, the request interval of $T_j$ that has a deadline in the interval $[t_i, t_d]$, must have started before $t_d$. Therefore, since $T_i$ has a deadline after $t_d$, the EDF algorithm will not choose $T_j$ before $T_i$ in the interval $[t_i, t_d]$. Hence, it is not possible for a task with index greater than $i$ to be scheduled in the interval $[t_i, t_d]$.

Claim 5: There cannot have been any idle time in the interval $[t_i, t_d]$.

Proof of claim: Assume the contrary, that is, there exists at least one idle period in the interval $[t_i, t_d]$. In order for this to be the case, the idle period clearly must have occurred after the execution request of $T_i$ that contains $t_d$ has completed execution. Let $t_0$ be the end of the last idle period that occurs in the interval $[t_i, t_d]$. We know that

$$t_s < t_i + p_i \leq t_0 \leq t_d.$$

From Claim 4 and the definition of $T_i$, we must have
\[ d_{t_0, t_d} \leq \sum_{j=1}^{i-1} \left\lfloor \frac{t_d - t_0}{p_j} \right\rfloor c_j \]

\[ = \sum_{j=1}^{n} \left\lfloor \frac{t_d - t_0}{p_j} \right\rfloor c_j \]

\[ \leq \sum_{j=1}^{n} \frac{t_d - t_0}{p_j} c_j \]

and

\[ t_d - t_0 < d_{t_0, t_d}. \]

This gives us

\[ t_d - t_0 < \sum_{j=1}^{n} \frac{t_d - t_0}{p_j} c_j \]

\[ < (t_d - t_0) \sum_{j=1}^{n} \frac{c_j}{p_j} , \]

which again implies that

\[ 1 < \sum_{j=1}^{n} \frac{c_j}{p_j}. \]

This is a contradiction of condition (1).

\[ \Delta \]

Returning to the proof of Case 2, we will show that if \( T_i \) is scheduled prior to \( t_d \), then there must have existed enough processor time in \([t_0, t_d] \) to execute \( T_k \). The first three claims above tell us that we have a pattern of execution requests in \([t_0, t_d] \) as shown below.
Since the request interval of $T_i$ scheduled at $t_i$ has a deadline after $t_d$, all outstanding requests for execution at $t_i$ with deadlines before $t_d$ must have been satisfied by $t_i$. In the above scenario, the execution requests eligible to be scheduled in $[t_i, t_d]$ are shaded in the figure below.

Hence using Claims 3 and 4, we can bound the total processor demand in $[t_i, t_d]$ by noting

$$d_{t_i, t_d} \leq c_i + \sum_{j=1}^{i-1} \left\lfloor \frac{(t_d - t_i) - 1}{p_j} \right\rfloor c_j.$$

Let $l = (t_d - t_i) - p_k$. Substituting $l$ into the above inequality we get
\[ d_{t_i,t_d} \leq c_i + \sum_{j=1}^{i-1} \left\lfloor \frac{p_k + l - 1}{p_j} \right\rfloor c_j. \]

Now from Claim 5 we know that there cannot be any idle time in \([t_i,t_d]\). Therefore, because a task missed a deadline at \(t_d\) we must have

\[ t_d - t_i < d_{t_i,t_d}, \]

or from the definition of \(l\),

\[ l + p_k < d_{t_i,t_d}. \]

Combining this with the previous inequality we have

\[ l + p_k < c_i + \sum_{j=1}^{i-1} \left\lfloor \frac{p_k + l - 1}{p_j} \right\rfloor c_j, \]

or simply

\[ p_k < c_i - l + \sum_{j=1}^{i-1} \left\lfloor \frac{p_k + l - 1}{p_j} \right\rfloor c_j. \]

Now since \(0 < l < p_i - p_k\), we have a contradiction of condition (2). Therefore the theorem is proved.

5. References

[Conway et al. 67]