An Analysis of EDF Schedulability on a Multiprocessor

Theodore P. Baker, *Senior Member*, *IEEE*

Abstract—A new schedulability test is derived for preemptive deadline scheduling of periodic or sporadic real-time tasks on a single-queue *m*-server system. The new test allows the task deadline to be more or less than the task period, and is based on a new analysis concept, called a μ-busy interval. This generalizes a result of Goossens et al. [11] that a system of periodic tasks with maximum individual task utilization u_{max} is EDF-schedulable on m processors if the total utilization does not exceed $m(1-u_{max})+u_{max}$. The new test allows the analysis of hybrid EDF-US [x] scheduling, and the conclusion that EDF-US[1/2] is optimal, with a guaranteed worst-case schedulable utilization of $(m+1)/2$.

Index Terms—Multiprocessor systems, real-time scheduling, deadline scheduling, earliest deadline first, utilization, feasibility, multiprocessor scheduling.

1 Introduction

It has long been known that preemptive earliest-deadline-first (EDF) scheduling is optimal for single-processor batch-oriented systems, in the sense that a set of independent jobs with deadlines is feasible if and only if it is EDF-schedulable. Liu and Layland [16] extended this result to systems of independent periodic tasks for which the relative deadline of each task is equal to its period, proving that so long as the total processing demand does not exceed 100 percent of the system capacity EDF will not permit any missed deadlines. Besides showing that that EDF scheduling is optimal for such task systems, this utilization bound provides a simple and effective schedulability test.

It also well-known that the optimality of EDF scheduling breaks down on multiprocessor systems [17]. For example, consider a batch of jobs $J_1,\ldots,J_{m+1}$ with deadlines $d_1=\cdots=d_m=mx$, $d_{m+1}=mx+1$, and computation times $c_1=\cdots=c_m=1$, $c_{m+1}=mx+1$, where x is an arbitrary constant, all released at time zero. If job J_{m+1} is allowed to monopolize one processor, all the tasks can be completed by their deadlines using m processors. However, an EDF scheduler for m processors would schedule jobs $J_1\ldots J_m$ ahead of J_{m+1}, causing J_{m+1} to miss its deadline.

Dhall and Liu [9] showed that the effect illustrated above means one cannot guarantee EDF schedulability on m processors for a utilization any higher than can be guaranteed on a single processor. It is tempting to conclude from Dhall and Liu's example that there is no useful utilization bound test for EDF scheduling, or that EDF is not a good real-time scheduling policy for multiprocessor systems. However, neither conclusion is actually justified.

Dhall and Liu's result depends on two extreme kinds of tasks: "heavy" ones, with a high ratio of computation time to deadline, and "light" ones, with a low ratio of computation time to deadline. It is the mixing of those two kinds of tasks that causes a problem for EDF. A modified EDF policy that segregates the heavy tasks from the light tasks, on disjoint sets of CPUs, would have no problem with this example. Examination of further examples leads one to conjecture that such a segregated EDF scheduling policy would not miss any deadlines until a very high level of CPU utilization is achieved, and may even permit the use of simple utilization-based schedulability tests.

Perhaps motivated by reasoning similar to that above, Srinivasan and Baruah [19] examined the deadline-based scheduling of periodic tasks on multiprocessors, and showed that any system of independent periodic tasks for which the utilization of every individual task is at most $m/(2m-1)$ can be scheduled successfully on m processors if the total utilization is at most $m^2/(2m-1)$. They then proposed a hybrid scheduling algorithm, called EDF-US[$m/(2m-1)$], that gives higher priority to tasks with utilizations above $m/(2m-1)$, and showed that EDF-US[$m/(2m-1)$] is able to successfully schedule *any* set of independent periodic tasks with total utilization up to $m^2/(2m-1)$. These results were generalized in [11], where it is shown that a system of independent periodic tasks can be scheduled successfully on m processors by EDF scheduling if the total utilization is at most $m(1-u_{max})+u_{max}$, where u_{max} is the maximum utilization of any individual task.

The above cited results were derived indirectly, using a theorem relating EDF schedulability on sets of processors of different speeds by Phillips et al. [18]. We have approached the problem more directly, and obtained a different and more general schedulability condition, of which the above cited results turn out to be special cases.

The rest of this paper presents the derivation of this more general multiprocessor EDF schedulability condition. Section 2 defines the problem formally, and outlines the overall approach. Section 3 derives a lower bound on the workload

• *The author is with the Department of Computer Science, Florida State University, 207A Love Building, Palmetto Drive, Tallahassee, FL 32306-4530. E-mail: baker@cs.fsu.edu*

Manuscript received 23 Dec. 2003; revised 13 June 2004; accepted 30 Oct. 2004; published online 22 June 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0238-1203.

contributions of competing tasks in an interval where a task misses a deadline. Section 4 defines the notion of μ-busy interval and derives an upper bound on the workload contributions of competing tasks in any such interval. Section 5 combines the upper and lower bounds on workload to obtain schedulability tests. Section 6 reviews in more detail the prior work mentioned above, shows how it relates to the new schedulability tests, and derives a new result regarding the optimality of EDF-US[1/2] scheduling. Section 7 summarizes and concludes.

2 Definition of the Problem

Suppose one is given a set of N simple independent periodic tasks $\tau_1, \ldots, \tau_N$, where each task τ_i has minimum interrelease time (called *period* for short) T_i, worst-case computation time c_i, and relative deadline d_i, where $c_i \leq d_i$ and $c_i \leq T_i$. Each task generates a sequence of *jobs*, each of whose release time is separated from that of its predecessor by at least T_i. (No special assumptions are made about the first release time of each task.)

Time is represented by rational numbers. Square brackets and parentheses are used to distinguish whether time intervals include their endpoints. For example, the time interval $[t_1, t_2)$ contains the time values greater than or equal to t_1 and less than t_2. All of the intervals $[t_1, t_2]$, $[t_1, t_2)$, $(t_1, t_2]$, and (t_1, t_2) are said to be of *length* $t_2 - t_1$.

Note that what we call a periodic task here is sometimes called a sporadic task. In this regard, we follow Liu [17], who observed that defining periodic tasks to have interrelease times exactly equal to the period "has led to the common misconception that scheduling and validation algorithms based on the periodic task model are applicable only when every periodic task is truly periodic...in fact, most existing results remain correct as long as interrelease times of jobs in each task are bounded from below by the period of the task."

In this paper, we assume that the jobs of a set of periodic tasks are scheduled preemptively according to an EDF policy on m processors, with dynamic processor assignment. That is, whenever there are m or fewer jobs ready they will all be executing, and whenever there are more than m jobs ready there will be m jobs executing, all with deadlines earlier than or equal to the deadlines of the jobs that are not executing.

Our objective is to formulate a simple test for schedulability, expressed in terms of the periods, deadlines, and worst-case computation times of the tasks, such that if the test is passed no deadlines will be missed. We approach the problem by analyzing the minimum processor demand that is needed over an interval of time to cause a missed deadline.

Definition 1 (demand). *The* demand *of a time interval is the total amount of computation that would need to be completed within the interval for all the deadlines within the interval to be met.*

Definition 2 (load). *The* load *of an interval is W/Δ, where W is the demand of the interval and Δ is the length of the interval.*

Definition 3 (first missed deadline). *Consider any sequence of job release times and computation times that are consistent with the interrelease and worst-case computation time constraints and produce a schedule with a missed deadline. The first point in such a schedule at which a deadline is missed is a* first missed deadline.

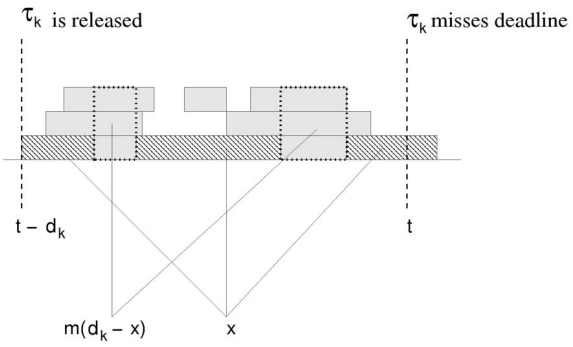


Fig. 1. All processors must be busy whenever τ_k is not executing.

Since T_i is only a lower bound on interrelease times and c_i is only an upper bound on execution times, and since the first start time of each task is also variable, there are infinitely many possible schedules. Each first missed deadline is understood to occur within the context of one of these schedules.

If we can find a lower bound on the processor load over an interval leading up to every first missed deadline, and we can guarantee that a given set of tasks could not possibly generate so much load in such an interval, that would be sufficient to serve as a proof of schedulability.

3 Lower Bound on Load

One can establish a lower bound on the load of the interval between a first missed deadline and the release time of the corresponding job by observing that, since the job does not complete by the end of the interval, the lengths of all the subintervals in which the job that misses its deadline does not execute must exceed its slack time. This fact is well known, and is used by Phillips et al. in [18]. It is illustrated Fig. 1, for the case where $m = 3$ and $d_k \leq T_k$. The diagonally shaded rectangles indicate blocks of time during which τ_k executes. The dotted rectangles indicate times during which all m processors must be busy executing other jobs that contribute to the demand for this interval. It is easy to see that the total demand of the interval $[t - d_k, t)$ must be at least $x + m(d_k - x)$, where $x \leq c_k$ is the amount of time that the the job of τ_k that misses its deadline executes in the interval.

To allow for the possibility that $d_k \geq T_k$, we need to extend the reasoning above to intervals that may include more than one job of the task that misses a deadline. Fig. 2 shows the release times and deadlines of two such jobs $\tau_{k,1}$ and $\tau_{k,2}$. The first job of τ_k is delayed by two blocks of higher priority interference. This interference is not enough to cause this job to miss its deadline, but (because jobs of the same task must be executed sequentially) it delays the start of the next job of τ_k enough to cause that job to miss its deadline.

Definition 4 (backlogged). *A job is* backlogged *at a time t if the task has a job that is released before time t and has nonzero execution time remaining at t.*

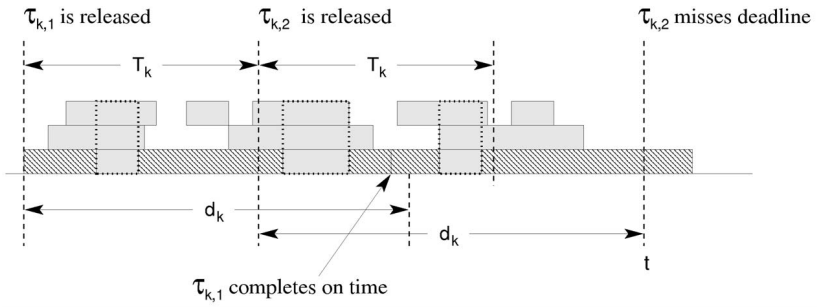


Fig. 2. More than one job of τ_k may execute in a τ_k-busy interval if $d_k \geq T_k$.

Definition 5 (τ_k-busy interval). *For any task τ_k, a time interval $[t', t)$ is τ_k-busy if there are backlogged jobs of τ_k continually throughout the interval (t', t).*

Lemma 1. *If a task τ_k is backlogged at time t then there is a unique maximal τ_k-busy interval $[t - \Delta, t)$, and there are no backlogged jobs of τ_k at time $t - \Delta$.*

Proof. Let t' be the release time of a job of τ_k that is backlogged at time t. The interval $[t', t)$ is τ_k-busy. If there is another job of τ_k that is backlogged at time t', the τ_k-busy interval can be extended downward. However, there is a limit to such extensions, since the system has some start time, before which no task is released. The set of all τ_k-busy intervals with endpoint t is totally ordered by length. Therefore, it has a unique maximal element, $[t - \Delta, t)$. □

Note that if $d_k \leq T_k$ the maximal τ_k-busy interval cannot be longer than d_k, but if $d_k > T_k$ the interval may be arbitrarily longer than d_k. For example, consider a system of $m + 1$ tasks, such that $T_1 = \cdots = T_{m+1} = d_1 = \cdots = d_m = 1$, $d_{m+1} = 2$, $c_1 = \cdots = c_m = \epsilon$, and $c_{m+1} = 1$. The first m tasks create a block of interference of duration ϵ for every release of τ_{m+1}, so that each successsive job of τ_{m+1} completes later by ϵ. The first job of task τ_{m+1} to miss its deadline will be the jth job, where j is the least integer greater than $\frac{1}{\epsilon}$. It will miss its deadline at time $jT_{m+1} + d_{m+1} = j + 1$, since $j\epsilon > d_{m+1} - T_{m+1} = 1$. We can make j arbitrarily large by choosing ϵ small enough.

Lemma 2 (lower bound on load). *If W is the demand of a maximal τ_k-busy interval $[t - \Delta, t)$, then*

$$\frac{W}{\Delta} > m - (m-1)\frac{c_k}{\min\{d_k, T_k\}}.$$

Proof. Let x be the amount of time τ_k executes in the interval $[t - \Delta, t)$. Since there are no backlogged jobs of τ_k at the start of the interval, the demand of τ_k in the interval consists only of jobs that are released in the interval. Let $j \geq 1$ be the number of such jobs that have deadlines on or before t. The demand of τ_k over the interval is jc_k. Since τ_k is backlogged continually over the interval, we know that $x < jc_k$.

Since the j jobs are all released on or after $t - \Delta$, have deadlines on or before $t - d_k$, and have release times separated by at least T_k, we know that

$$\begin{aligned}(j-1)T_k + d_k &\leq \Delta \\ j &\leq \frac{\Delta + T_k - d_k}{T_k} \\ x < jc_k &\leq \frac{c_k}{T_k}(\Delta + T_k - d_k).\end{aligned}$$

Since a processor is never idle while a job is waiting to execute, during every subinterval in which τ_k is not executing all m processors are busy executing jobs of other tasks. Let y be the sum of the lengths of all the subintervals for which all m processors are executing jobs of tasks other than τ_k. The entire demand of the interval $[t - \Delta, t)$ must be at least $x + my$. It follows that

$$\begin{aligned}W &\geq x + my = x + m(\Delta - x) = m\Delta - x(m-1) \\ &> m\Delta - (m-1)\frac{c_k}{T_k}(\Delta + T_k - d_k).\end{aligned}$$

Dividing both sides of the above inequality by Δ, we obtain

$$\frac{W}{\Delta} > m - (m-1)\frac{c_k}{T_k}\left(1 + \frac{T_k - d_k}{\Delta}\right).$$

If $d_k \leq T_k$, we can use the fact that $\Delta > d_k$ and obtain

$$\frac{W}{\Delta} > m - (m-1)\frac{c_k}{d_k}.$$

Otherwise, if $d_k > T_k$, we can take the limit as $\Delta \to \infty$ and obtain

$$\frac{W}{\Delta} > m - (m-1)\frac{c_k}{T_k}.$$

□

4 Upper Bound on Load

We now derive an upper bound on the load $\frac{W}{\Delta}$ of an interval leading up to a first missed deadline. If we can find such an upper bound β, it will follow that schedulability of a task system can be guaranteed by checking that β is less than the minimum load needed to cause a missed deadline. The upper bound β will be defined as the sum of individual upper bounds $\beta_k(i)$ on the *load contribution* W_i/Δ for each individual task in the interval. The contribution W_i of each task τ_i to the demand of such an interval consists of two parts:

1. The computation time of jobs of τ_i that are released within the interval.

2. The computation time of jobs of τ_i that are released before the start of the interval, which we call the *carried-in* contribution of τ_i.

To bound the size of the carried-in contribution of τ_i, we extend the notion of τ_k-busy interval as follows:

Definition 6 (μ-busy interval). *A time interval is* μ-busy *if its total load is at least* μ. *A* downward extension *of a* μ-busy *interval is the interval itself or a* μ-busy *interval that has an earlier starting point and shares the same endpoint. A* proper *downward extension of a* μ-busy *interval is a downward extension that has an earlier starting point. A* maximal μ-busy *downward extension of a* μ-busy *interval is a downward extension that has no further proper downward extensions.*

Lemma 3. *Any* τ_k-busy *interval* $[t-\Delta, t)$ *has a unique maximal* μ-busy *downward extension* $[t-\hat{\Delta}, t)$ *for any* $\mu \leq m-(m-1)\frac{c_k}{\min\{d_k,T_k\}}$.

Proof. The proof is analogous to that of Lemma 1. Let $[t-\Delta, t)$ be any τ_k-busy interval. By Lemma 2, this interval is μ-busy, so the set of μ-busy downward extensions of $[t-\Delta, t)$ is nonempty. The system has some start time, before which no task is released, so the length of all μ-busy downward extensions of any such interval is bounded. Any set of intervals with a common endpoint is totally ordered by length. Therefore, the set of μ-busy downward extensions of $[t-\Delta, t)$ has a unique maximal element. □

Definition 7 (maximal μ-busy interval). *For any first missed deadline* t *of a system, if* τ_k *is a task that misses a deadline at time* t *and* $\mu \leq m-(m-1)\frac{c_k}{\min\{d_k,T_k\}}$, *then Lemma 1 guarantees that there is a unique maximal* τ_k-busy *interval* $[t-\Delta, t)$, *and Lemma 3 guarantees that it has a unique maximal* μ-busy *downward extension* $[t-\hat{\Delta}, t)$, *such that* $\hat{\Delta} \geq \Delta \geq d_k$. *We call the interval* $[t-\hat{\Delta}, t)$ *the* maximal μ-busy interval *of the first missed deadline* t.

We next look for an upper bound on the contribution W_i of each task τ_i to the demand W of a maximal μ-busy interval. The analysis is divided into cases, according to whether τ_i makes any carried-in contribution.

Case 1. If there is no carried-in contribution, the analysis is simple. No jobs released prior to $t-\hat{\Delta}$ can contribute to W_i. Therefore, an upper bound on W_i can be obtained by multiplying the worst-case computation time of τ_i by the maximum number of jobs of τ_i that may be released in an interval of length $\hat{\Delta}-d_i$.

$$W_i \leq \max\left\{0, c_i\left\lceil\frac{\hat{\Delta}-d_i}{T_i}\right\rceil\right\} \leq \max\left\{0, \frac{c_i}{T_i}(\hat{\Delta}+T_i-d_i)\right\}.$$

Since $\hat{\Delta} \geq d_k$, it follows that

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i-d_i}{d_k}\right) \text{ if } d_i \leq T_i, \text{ and} \quad (1)$$

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i} \text{ if } d_i > T_i. \quad (2)$$

It now only remains to analyze the case where there is a carried-in contribution. Therefore, from this point on, we assume there is at least one job of τ_i backlogged at time $t-\hat{\Delta}$.

Definition 8 (preamble). *By Lemma 1, if* $[t-\hat{\Delta}, t)$ *is a maximal* μ-busy *interval and there is a job of* τ_i *backlogged at time* $t-\hat{\Delta}$, *then there is a unique maximal* τ_i-busy *interval* $[t-\hat{\Delta}-\phi, t-\hat{\Delta})$, *for some* $\phi > 0$. *We call this interval the* preamble *with respect to* τ_i *of the maximal* μ-busy *interval* $[t-\hat{\Delta}, t)$.

It follows that:

1. There are no backlogged jobs of τ_i at time $t-\hat{\Delta}-\phi$.
2. A job of τ_i is released at time $t-\hat{\Delta}-\phi$.
3. At every point in the interval $(t-\hat{\Delta}-\phi, t-\hat{\Delta})$, there is at least one backlogged job of τ_i.

Lemma 4 (upper bound on carry-in). *If* $[t-\hat{\Delta}-\phi, t-\hat{\Delta})$ *is the preamble of* $[t-\hat{\Delta}, t)$ *with respect to task* τ_i, *then the amount* x *of computation that the task* τ_i *completes in the preamble is greater than* $\lambda\phi$, *where* $\lambda = \frac{m-\mu}{m-1}$.

Proof. The reasoning is like that of Lemma 2. Let x be the total amount of time spent executing jobs of τ_i in the preamble and let y be the sum of the lengths of all the subintervals within the preamble where all m processors are simultaneously executing jobs that preempt τ_i. Since τ_i can execute when and only when there are less than m processors executing jobs that preempt τ_i, we know that $y = \phi - x$. It follows that the total amount of work executed in the preamble on τ_i must be at least $my + x$. Putting this together with the fact that the interval $[t-\hat{\Delta}, t)$ is μ-busy, one can conclude that the load of the combined interval $[t-\hat{\Delta}-\phi, t)$ is at least $\mu\hat{\Delta} + my + x$.

Since $[t-\hat{\Delta}, t)$ has no proper μ-busy downward extension, the demand of the interval $[t-\hat{\Delta}-\phi, t)$ is less than $\mu(\phi+\hat{\Delta})$. It follows that

$$\begin{aligned}
\mu\hat{\Delta} + my + x &< \mu(\phi+\hat{\Delta}) \\
my + x &< \mu\phi \\
m(\phi - x) + x &< (m - \lambda(m-1))\phi \\
m\phi - x(m-1) &< m\phi - \lambda(m-1)\phi \\
x(m-1) &> \lambda(m-1)\phi \\
x &> \lambda\phi.
\end{aligned}$$

□

Let j be the number of jobs of τ_i released in the preamble. Since, by Lemma 4, τ_i executes for at least $\lambda\phi$ time in the preamble, the carried-in contribution of τ_i to W_i is less than $jc_i - \lambda\phi$.

The noncarried-in portion of W_i comes from jobs released in the interval $[t-\hat{\Delta}-\phi+jT_i, t-d_i]$. This contribution is bounded by $c_i\lceil\frac{\hat{\Delta}+\phi-jT_i-d_i}{T_i}\rceil$. (We are assuming τ_i makes a nonzero carried-in contribution, so this quantity must be positive.) Combining the upper bounds on the carried-in and noncarried-in contributions to W_i, we have

$$W_i < jc_i - \lambda\phi + c_i\left\lceil\frac{\hat{\Delta}+\phi-jT_i-d_i}{T_i}\right\rceil$$
$$= c_i\left\lceil\frac{\hat{\Delta}+\phi-d_i}{T_i}\right\rceil - \lambda\phi$$
$$\le c_i\left(\frac{\hat{\Delta}+\phi-d_i}{T_i}+1\right) - \lambda\phi.$$

That is,

$$W_i < \frac{c_i}{T_i}(\hat{\Delta}+T_i-d_i) + \phi\left(\frac{c_i}{T_i}-\lambda\right). \quad (3)$$

We will consider separately the case where $\frac{c_i}{T_i} \le \lambda$ and the case where $\frac{c_i}{T_i} > \lambda$.

Case 2.1. If $\frac{c_i}{T_i} \le \lambda$, the value of the expression on the right of (3) is nonincreasing with respect to ϕ, and since $\phi \ge 0$, the global maximum is achieved when $\phi = 0$. It follows that the upper bounds (1) and (2) for $\frac{W_i}{\hat{\Delta}}$ derived in Case 1 also apply in this case.

Case 2.2. If $\frac{c_i}{T_i} > \lambda$, the value of the expression on the right of (3) is increasing with respect to ϕ. We will consider separately the case where $d_i \le T_i$ and the case where $d_i > T_i$.

Case 2.2.1. If $d_i \le T_i$, each job of τ_i must complete by the release time of the next job. It follows that there can be no τ_i-busy interval of length longer than d_i, and so $\phi < d_i$. Using this fact and that the expression on the right of (3) is increasing with respect to ϕ, we have

$$W_i < \frac{c_i}{T_i}(\hat{\Delta}+T_i-d_i) + d_i\left(\frac{c_i}{T_i}-\lambda\right).$$

Since $c_i - \lambda T_i > 0$ and $\hat{\Delta} \ge d_k$, it follows that

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i}{d_k}\right) - \lambda\frac{d_i}{d_k}. \quad (4)$$

Observe that the bound above is greater than or equal to the corresponding bound (1) derived for Case 1.

Case 2.2.2. If $d_i > T_i$, then ϕ may be arbitrarily large, and so inequality (3) is not useful; we need to find another way to bound W_i. Let ψ be the offset from the start of the maximal τ_i-busy interval $[t-\hat{\Delta}, t)$ of the release time of the first carried-in job of τ_i. That is, $t-\hat{\Delta}-\psi$ is the first time before $t-\hat{\Delta}$ where a job of τ_i is released that is still backlogged at time $t-\hat{\Delta}$. All the jobs of τ_i that contribute to the demand of $[t-\hat{\Delta}, t)$ must be released within the interval $[t-\hat{\Delta}-\psi, t-d_i]$. The maximum number of jobs of τ_i that can be released in an interval of this length an have a deadline in the interval is

$$\left\lceil\frac{\hat{\Delta}+\psi-d_i}{T_i}\right\rceil. \quad (5)$$

The job of τ_i released at time $t-\hat{\Delta}-\psi$ is still backlogged at time $t-\hat{\Delta}$, but since the first missed deadline comes later, at time t, this job must complete by its deadline, which is $t-\hat{\Delta}-\psi+d_i$. It follows that $\psi < d_i$. Putting this upper bound on ψ together with the upper bound (5) on the number of jobs that contribute to W_i, and multiplying by the worst-case computation time of τ_i, we obtain

$$W_i \le c_i\left\lceil\frac{\hat{\Delta}-d_i+\psi}{T_i}\right\rceil < c_i\left\lceil\frac{\hat{\Delta}}{T_i}\right\rceil \le \frac{c_i}{T_i}(\hat{\Delta}+T_i).$$

Since $\hat{\Delta} > d_k$, it follows that

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i}{d_k}\right). \quad (6)$$

Observe that the bound above is greater than or equal to the corresponding bound (2) derived for Case 1.

TABLE 1
Cases for $\beta_k(i)$, the Upper Bound on $\frac{W_i}{\Delta}$

	$\frac{c_i}{T_i} \le \lambda$	$\frac{c_i}{T_i} > \lambda$
$d_i \le T_i$	$\frac{c_i}{T_i}(1+\frac{T_i-d_i}{d_k})$	$\frac{c_i}{T_i}(1+\frac{T_i}{d_k}) - \lambda\frac{d_i}{d_k}$
$d_i > T_i$	$\frac{c_i}{T_i}$	$\frac{c_i}{T_i}(1+\frac{T_i}{d_k})$

Lemma 5 (upper bounds on load). *If $[t-\hat{\Delta}, t)$ is a maximal μ-busy interval for task τ_k, then the contribution $\frac{W_i}{\hat{\Delta}}$ of each task τ_i to the demand of the interval is strictly bounded above by the function $\beta_k(i)$ defined by Table 1, where $\lambda = \frac{m-\mu}{m-1}$.*

Proof. We showed in the analysis for Cases 1 and 2.1 above that if $\frac{c_i}{T_i} \le \lambda$ and $d_i \le T_i$, then

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i-d_i}{d_k}\right),$$

and if $\frac{c_i}{T_i} \le \lambda$ and $d_i > T_i$, then

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}.$$

We showed in the analysis for Cases 1 and 2.2 above that if $\frac{c_i}{T_i} > \lambda$ and $d_i \le T_i$, then

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i}{d_k}\right) - \lambda\frac{d_i}{d_k},$$

and if $\frac{c_i}{T_i} > \lambda$ and $d_i > T_i$, then

$$\frac{W_i}{\hat{\Delta}} < \frac{c_i}{T_i}\left(1+\frac{T_i}{d_k}\right).$$

□

5 Schedulability Condition

Using the above analysis, we can now prove the following theorem, which provides a sufficient condition for EDF schedulability.

Theorem 1 (EDF schedulability test). *A set of periodic tasks $\tau_1, \ldots, \tau_N$ is schedulable on m processors using preemptive EDF scheduling if, for every task τ_k, there exists a positive value $\mu \le m - (m-1)\frac{c_k}{\min\{d_k, T_k\}}$ such that*

$$\sum_{i=1}^{N} \beta_k(i) \le \mu, \quad (7)$$

where $\lambda = \frac{m-\mu}{m-1}$ and $\beta_k(i)$ is as defined in Table 1.

Proof. The proof is by contradiction. Suppose some task misses a deadline. Let τ_k be any first task to miss a deadline. (If more than one misses a deadline together at the same time, choose any one of them.) Let $[t-\hat{\Delta}, t)$ be the maximal μ-busy interval guaranteed by Lemma 1 and Lemma 3. Since $(t-\Delta, t)$ is μ-busy, we have

$$\frac{W}{\hat{\Delta}} > \mu.$$

By Lemma 5, $\frac{W_i}{\hat{\Delta}} < \beta_k(i)$, for $i = 1, \ldots, N$, and so

$$\sum_{i=1}^{N} \beta_k(i) > \frac{W}{\hat{\Delta}} > \mu.$$

The above is a contradiction of (7). □

To get the most accuracy from the above condition as a schedulability test, it might seem necessary to consider all possible values of μ for each k. However, the only values of μ that need to be considered are the upper bound and the points at which $\beta_k(i)$ is discontinuous with respect to the implicit parameter μ. That is, at the points

$$\mu_i = m - \frac{c_i}{T_i}(m-1)$$

for $i = 1, \ldots, k$, and

$$\mu_{\max} = m - \frac{c_k}{\min\{d_k, t_k\}}(m-1).$$

The schedulability test above must be checked individually for each task τ_k. If we are willing to sacrifice some precision, there is a simpler test that only needs to be checked once for the entire system of tasks.

Corollary 1 (simplified test). *A set of periodic tasks $\tau_1, \ldots, \tau_N$ is schedulable on m processors using preemptive EDF scheduling if*

$$\sum_{i=1}^{N} \frac{c_i}{T_i}\left(1 + \frac{\max\{0, T_i - d_i\}}{d_{min}}\right) \le m - \hat{\lambda}(m-1), \qquad (8)$$

where

$$\hat{\lambda} = \max\left\{\frac{c_i}{\min\{d_i, T_i\}} \mid i = 1, \ldots, N\right\}$$

and $d_{min} = \min\{d_k \mid i = 1, \ldots, N\}$.

Proof. Corollary 1 is proved by repeating the proof of Theorem 1, adapted to fit the definition of $\hat{\lambda}$. Let τ_k be a first task to miss a deadline. Let $\mu' = m - \hat{\lambda}(m-1)$. Since $\hat{\lambda} \ge \frac{c_k}{\min\{d_k, T_k\}}$, it follows that $\mu' \le m - (m-1)\frac{c_k}{\min\{d_k, T_k\}}$. Therefore, there is a maximal μ'-busy interval $[t-\hat{\Delta}, t)$ whose existence is guaranteed by Lemma 3. Since $[t-\hat{\Delta}, t)$ is μ'-busy, we have

$$\frac{W}{\hat{\Delta}} > m - \hat{\lambda}(m-1).$$

By Lemma 5, $\frac{W_i}{\hat{\Delta}} \le \beta_k(i)$, for $i = 1, \ldots, N$. Since $\hat{\lambda} \ge \frac{c_i}{\min\{d_i, T_i\}} \ge \frac{c_i}{T_i}$, only the first column of the definition of $\beta_k(i)$ in Table 1 applies, i.e.,

$$\beta_k(i) = \frac{c_i}{T_i}\left(1 + \frac{\max\{0, T_i - d_i\}}{d_k}\right).$$

It follows that

$$\begin{aligned}\sum_{i=1}^{N} \frac{c_i}{T_i}\left(1 + \frac{\max\{0, T_i - d_i\}}{d_{min}}\right) &\ge \sum_{i=1}^{N} \beta_k(i) \\ &\ge \frac{W}{\hat{\Delta}} \\ &> m - \hat{\lambda}(m-1).\end{aligned}$$

The above contradicts (8). □

If we replace d_i by T_i in Corollary 1, we immediately obtain an independent proof of the following utilization bound test, which was first reported in [11].

Theorem 2 (Goossens, Funk, and Baruah). *A set of periodic tasks $\tau_1, \ldots, \tau_N$, all with deadline equal to period, and with maximum individual task utilization u_{max} is guaranteed to be schedulable on m processors using preemptive EDF scheduling if the total system utilization U does not exceed $m - (m-1)u_{max}$.*

The results above are useful as schedulability tests. They can be applied directly to prove that a task set will meet deadlines with EDF scheduling, either before runtime for a fixed set of tasks or during runtime as an admission test for a system with a dynamic set of tasks. With the simpler forms, one checks the schedulability condition once for the entire task set. This computation can be performed in $\mathcal{O}(n)$ time. With the more general form, one checks the schedulability condition for each task and each of n possible values μ. This computation can be performed in $\mathcal{O}(n^3)$ time. In the latter case the specific value(s) of k for which the test fails provide some indication of where the problem lies.

The schedulability test of Theorem 1 allows preperiod deadlines (as well as postperiod deadlines), but it is more complicated than the utilization bound test. One may naturally wonder whether this extra complexity gains anything over the well-known technique for handling preperiod deadlines by "padding" computation times and then using the utilization bound test. By padding the computation times, we mean that if a task τ_k has computation time c_k and deadline $d_k < T_k$, we replace it (for the purpose of schedulability testing only) by τ'_k, where $c'_k = c_k + T_k - d_k$ and $d'_k = T'_k = T_k$. If τ'_k will always complete within its period, then τ_k will always complete $T_k - d_k$ time units before its period. That is, with EDF scheduling, the original task τ_k can be scheduled to meet its deadline if the following condition holds for τ'_k.

$$\frac{T_k - d_k}{T_k} + U \le m(1 - u'_{max}) + u'_{max}, \qquad (9)$$

where

$$u'_{max} = \max\left\{\frac{c_i}{T_i} \mid i = 1, \ldots, N\right\} \cup \left\{\frac{c'_i}{T_i}\right\}$$

and $U = \sum_{i=1}^{N} \frac{c_i}{T_i}$.

To compare the accuracy of these two tests, consider the case where just task τ_k has a preperiod deadline, $d_k < T_k$. For

simplicity, consider just the value $\mu = m - \hat{\lambda}(m-1)$, where $\hat{\lambda}$ is as in the proof of Corollary 1. For task τ_k, $\beta_k(k) = \frac{c_k}{T_k} + \frac{T_k - d_k}{T_k}\frac{c_k}{d_k}$. For all the other tasks τ_i, $\beta_k(i) = \frac{c_i}{T_i}$. Therefore, the test of Theorem 1 for the case of τ_k reduces to

$$\frac{T_k - d_k}{T_k}\frac{c_k}{d_k} + U \leq m\left(1 - \frac{c_k}{d_k}\right) + \frac{c_k}{d_k}, \tag{10}$$

and if we assume $u'_{max} = \frac{c_k}{d_k} \geq u_{max}$, the padded utilization test (9) reduces to

$$\frac{T_k - d_k}{T_k} + U \leq m\left(1 - \frac{c_k}{d_k}\right) + \frac{c_k}{d_k}. \tag{11}$$

It is easy to see that (10) is more accurate than (11) if $c_k < d_k$.

For example, suppose we have three processors and six tasks, with periods $T_1 = \cdots = T_6 = 1$, computation times $c_1 = \cdots = c_6 = 1/3$, and deadlines $d_1 = \cdots = d_5 = 1$ and $d_6 = 2/3$. The system utilization is $U = 2$. If we apply the padded utilization test to τ_3, the padded computation time is $c'_6 = \frac{1}{3} + \left(1 - \frac{2}{3}\right) = \frac{2}{3}$, $u'_{max} = \frac{2}{3}$, and (9) fails:

$$\frac{1 - \frac{2}{3}}{1} + U > 2.33 > 1.667 > 3\left(1 - \frac{2}{3}\right) + \frac{2}{3}.$$

On the other hand, (10) is satisfied:

$$\frac{1 - \frac{2}{3}}{1}\cdot\frac{\frac{1}{3}}{\frac{2}{3}} + U < 2.167 < 2.33 < 3\left(1 - \frac{1}{3}\right) + \frac{1}{3}.$$

Of course, these schedulability tests are only *sufficient* conditions for schedulability. They are based on very conservative, worst-case assumptions about task periods and task phasing. However, the Liu and Layland $n(2^{1/n} - 1)$ utilization bound for rate monotonic scheduling is proof that a similarly conservative test can still be very useful.

None of the schedulability tests presented here should be used when $N \leq m$. In this case a system will always be schedulable up to a total utilization of m, but the schedulability tests will not recognize that. However, as soon as $N = m + 1$ the worst-case utilization bound becomes 1, as shown by Dhall and Liu, and the tests given here become useful.

6 Relation to Prior Work

Srinivasan and Baruah [19] defined a periodic task set $\{\tau_1, \tau_2, \ldots \tau_N\}$ to be a *light system on m processors* if it satisfies the following properties:

1. $\sum_{i=1}^{N} \frac{c_i}{T_i} \leq \frac{m^2}{2m-1}$.
2. $\frac{c_i}{T_i} \leq \frac{m}{2m-1}$, for $1 \leq i \leq N$.

They then proved that any periodic task system that is light on m processors is scheduled to meet all deadlines on m processors by EDF.

The above analysis was refined in [11], to obtain the $m(1 - u_{max}) + u_{max}$ utilization bound. In the same paper this utilization bound was shown to be tight, i.e., there is no utilization bound $\hat{U} > m(1 - u_{max}) + u_{max} + \epsilon$, where $\epsilon > 0$ and $u_{max} = \max\{c_i/T_i \mid i = 1, \ldots, N\}$, for which $U \leq \hat{U}$ guarantees EDF schedulability.

We have gone beyond the above cited result by covering the cases where the deadline of task is more or less than its period. Preperiod deadlines are useful for modeling tasks with bounded jitter requirements. They can also be useful for tasks that have precedence constraints, such as a first part that initiates input, and a second part that processes the resulting input. Postperiod deadlines are useful for modeling tasks that use buffers to cope with bursty inputs.

We also have provided a more direct proof of the $m(1 - u_{max}) + u_{max}$ utilization bound. The proof in [11] is derived from a theorem in [10], on scheduling for uniform multiprocessors, which in turn is based on a multiprocessor speed-up result by Phillips et al. [18]. The same utilization bound follows directly from Corollary 1.

Srinivasan and Baruah [19], proposed adapting their utilization-bound schedulability condition to situations where there are a few high-utilization tasks as follows:

Algorithm 1 ($EDF\text{-}US[m/(2m-1)]$):
(heavy task rule) If $c_i/T_i > m/(2m-1)$, then schedule τ_i's jobs at maximum priority (i.e., as if they had a deadline long in the past).
(light task rule) If $c_i/T_i \leq m/(2m-1)$ then schedule τ_i's jobs according to their normal deadlines.

They then proved that Algorithm EDF-US$[m/(2m-1)]$ correctly schedules on m processors any periodic task system whose utilization is at most $m^2/(2m-1)$. Their proof is based on the observation that their upper bound on total utilization guarantees that the number of heavy tasks cannot exceed m. The essence of the argument is that Algorithm EDF-US$[m/(2m-1)]$ can do no worse than scheduling each of the heavy tasks on its own processor, and then scheduling the remainder (which must must be light on the remaining processors) using EDF. The EDF-US$[m/(2m-1)]$ idea was further studied by [1], in the context of a variable task set, where the number of heavy tasks may vary over time. The $m(1 - u_{max}) + u_{max}$ utilization bound naturally suggests the following generalization of the EDF-US idea, for an arbitrary single-task utilization cut-off ζ.

Algorithm 2 ($EDF\text{-}US[\zeta]$) :
(heavy task rule) If $c_i/T_i > \zeta$, then schedule τ_i's jobs at maximum priority.
(light task rule) If $c_i/T_i \leq \zeta$, then schedule τ_i's jobs according to their normal deadlines.

Theorem 3. *Given a set of N periodic tasks, let k be the larger of $m - 1$ or the number of tasks with utilization higher than ζ, i.e., $k = \max\{m - 1, |\{i \mid u_i > \zeta\}|\}$.*

Algorithm EDF-US$[\zeta]$ *correctly schedules the task set on m processors if the combined utilization of the $N - k$ lightest tasks is at most $(m - k)(1 - \zeta) + \zeta$.*

Proof. As argued by Srinivasan and Baruah, if k is the number of heavy tasks (and $k \leq m - 1$), the performance of this algorithm cannot be worse than an algorithm that dedicates one processor to each of the k heavy tasks, and schedules the remaining $N - k$ tasks on the $m - k$ remaining processors. Theorem 2 guarantees the remaining $N - k$ tasks can be scheduled on

the remaining $m-k$ processors since their combined utilization is at most $(m-k)(1-\zeta)+\zeta$.

The remaining case is where there are more than k heavy tasks (and $k=m-1$). In this case, the performance of this algorithm cannot be worst than an algorithm that dedicates one processor to each of the k heaviest tasks and schedules the remaining $N-k$ tasks on the remaining single processor. Since the combined utilization of the $N-k$ lightest tasks is at most $(m-k)(1-\zeta)+\zeta=1$, they can all be scheduled on one processor. □

Observe that if a system is not guaranteed schedulable by Theorem 3, the total utilization must be greater than $k\zeta+(m-k)-((m-k)-1)\zeta$. Minimizing this expression with respect to k and maximizing with respect to ζ, we can see that the maximum value is achieved for $\zeta=1/2$. That is, Algorithm EDF-US[ζ] has the maximum guaranteed total schedulable utilization for $\zeta=1/2$.

Corollary 2. *Any task system is schedulable correctly by EDF-US[1/2] on m processors if the total utilization does not exceed $(m+1)/2$.*

Proof. Suppose the total utilization is U, $U\leq(m+1)/2$, and k is as in Theorem 3. We know the system is schedulable if the total utilization does not exceed

$$k/2+(m-k)-((m-k))-1)/2=$$
$$(k+2m-2k-m+k+1)/2=(m+1)/2.$$

□

The above bound is tight. This can be shown using the same argument used in [1] and [2] to show that "for all studied static-priority scheduling approaches (partitioning, global, and pfair global)" one cannot do better than utilization of $m/2$ on m processors. First, observe that even with deadline scheduling the priority for each job is fixed once the job is released. Consider a set of identical tasks $\tau_1,\ldots,\tau_{m+1}$ with $c_i=1/2+1/x$ and $d_i=T_i=1$. All the tasks have the same period, so if they are released together at the same time one job of each task will be competing with one job of the rest and all will have the same deadline. Any scheduling scheme that is based on assigning a fixed priority to each job will run m of the jobs until they have completed, and then run the remaining job. It will not have enough time to complete within its deadline. By choosing x large enough, one can make $U=(m+1)(1/2+1/x)$ arbitrarily close to $(m+1)/2$.

If there is a need to support preperiod and/or postperiod deadlines, the idea of singling out a few "ugly" tasks for higher priority treatment, which is at the core of the EDF-US algorithms, can be adapted further. Suppose one has m processors and a set of N tasks for which the schedulability test (that of Theorem 1 or, alternatively, that of Corollary 1) fails. One can try partitioning the N tasks into two sets: a set of k ($k<m$) ugly tasks to receive maximum priority, and schedule the remaining $N-k$ tasks using EDF scheduling. If the EDF schedulability test succeeds for the $N-k$ nice tasks on $m-k$ processors, then the entire task set is schedulable. A logical sequence of ugly task sets to try would be k tasks with the largest values of $\frac{c_i}{d_i}$, for $k=1,2,\ldots m-1$.

7 Conclusion

We have demonstrated an efficiently computable schedulability test for EDF scheduling on a homogeneous multiprocessor system, which allows preperiod and postperiod deadlines. It can be applied statically, or applied dynamically as an admission test. Besides extending and generalizing previously known utilization-based tests for EDF multiprocessor schedulability by supporting preperiod and postperiod deadlines, we have provided a distinct and independent proof using a new technique. We have also shown that EDF-US[1/2] is an optimal multiprocessor scheduling technique, with respect to maximizing the processor utilization at which deadlines can be missed.

The notion of μ-busy interval, used to derive an EDF schedulability condition here, has broader applications. In [6] and [5], the same approach is applied to the analysis of deadline monotonic scheduling for multiprocessor systems. The technique may also be applicable to more general multiprocessor systems, such as those studied in [10], or to probabilistic analysis.

Acknowledgments

The author is thankful to the anonymous reviewers for their constructive comments, which improved the quality of this paper. He is especially thankful to the reviewer who noticed a false statement in one of the proofs of the original paper, and the one reviewer who suggested extending the analysis to include postperiod deadlines. Some of the results reported here also appeared in a paper presented at the 2003 IEEE Real-Time Systems Symposium [5]. This paper differs from that paper by focusing specifically on deadline scheduling, including new and more detailed proofs, extending the analysis to include tasks with postperiod deadlines, and showing that EDF-US[1/2] is optimal.

References

[1] B. Andersson, "Static-Priority Scheduling on Multiprocessors," PhD thesis, Dept. of Computer Eng., Chalmers Univ. of Technology, Göteborg, Sweden, 2003.

[2] B. Andersson, S. Baruah, J. Jonsson, "Static-Priority Scheduling on Multiprocessors," *Proc. 21st IEEE Real-Time Systems Symp.*, pp. 193-202, Dec. 2001.

[3] J. Anderson and A. Srinivasan, "Early-Release Fair Scheduling," *Proc. 12th Euromicro Conf. Real-Time Systems*, pp. 34-43, June 2001.

[4] T.P. Baker, "Stack-Based Scheduling of Real-Time Processes," *The Real-Time Systems J.*, vol. 3, no. 1, pp. 67-100, Mar. 1991.

[5] T.P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis," *Proc. 23rd IEEE Real-Time Systems Symp.*, Dec. 2003.

[6] T.P. Baker, "An Analysis of Deadline-Monotonic Scheduling on a Multiprocessor," Florida State Univ., Dept. of Computer Science, technical report, 2003.

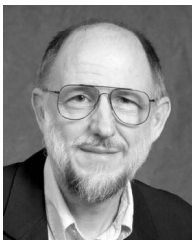
[7] S. Baruah, N. Cohen, C.G. Plaxton, D. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica 15*, pp. 600-625, 1996.

[8] S. Baruah, J. Gherke, C.G. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proc. Ninth Int'l Parallel Processing Symp.*, pp. 280-288, Apr. 1995.

[9] S.K. Dhall and C.L. Liu, "On a Real-Time Scheduling Problem," *Operations Research 26*, vol. 1, pp. 127-140, 1978.

[10] S. Funk, J. Goossens, S. Baruah, "On-Line Scheduling on Uniform Multiprocessors," *Proc. 21st IEEE Real-Time Systems Symp.*, pp. 183-192, Dec. 2001.

[11] J. Goossens, S. Funk, and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," *Real Time Systems,* vol. 25, nos. 2/3, pp. 187-205, Kluwer, Sept./Nov. 2003.
[12] T.M. Ghazalie and T.P. Baker, "Aperiodic Servers in a Deadline Scheduling Environment," *Real-Time Systems J.,* vol. 9, no. 1, pp. 31-68, Kluwer, July 1995.
[13] R. Ha, "Validating Timing Constraints in Multiprocessor and Distributed Systems," PhD thesis, Technical Report UIUCDCS-R-95-1907, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1995.
[14] R. Ha and J.W. S. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems," Technical Report UIUCDCS-R-93-1833, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Oct. 1993.
[15] R. Ha and J.W. S. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems," *Proc. 14th IEEE Int'l Conf. Distributed Computing Systems,* June 1994.
[16] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.
[17] J.W. S. Liu, *Real-Time Systems.* Prentice-Hall, 2000.
[18] C.A. Phillips, C. Stein, E. Torng, and J Wein, "Optimal Time-Critical Scheduling via Resource Augmentation," *Proc. 29th Ann. ACM Symp. Theory of Computing,* pp. 140-149, 1997.
[19] A. Srinivasan and S. Baruah, "Deadline-Based Scheduling of Periodic Task Systems on Multiprocessors," *Information Processing Letters,* vol. 84, pp. 93-98, 2002.



Theodore P. Baker received the PhD degree in computer science from Cornell University in 1974, for research on relative computability and computational complexity. He is a professor in the Department of Computer Science at Florida State University (FSU), and served four years as chair of that department. Starting in 1979, Professor Baker became involved with the development of the Ada programming language. The group he organized at FSU produced one of the first validated Ada cross-compilers for embedded systems. Since then, he has done research, development, and consulting related to real-time embedded computing, from basic research on scheduling and concurrency control through development of kernels and runtime system support for real-time programming languages. He has also been active in IEEE (POSIX) and ISO standards work related to real-time systems. He was a member of the SEI Rate Monotonic Analysis group, served as real-time area expert for the Ada 9X language mapping and revision team, and was a member of the 1997 National Research Council panel on Software Policies for the Department of Defense. He directed the FSU teams that developed several software artifacts, including the FSU POSIX threads library, the Florist implementation of the POSIX.5 API, a validation suite for the same, and the multitasking runtime system for the Gnu Ada (GNAT) compiler. He directed the porting of the latter to several environments, including the Java Virtual Machine and RTLinux. Professor Baker's current research focii are multiprocessor scheduling theory and experimentation, and real-time device driver architecture. He is a senior member of the IEEE Computer Society.