



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction

[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

Tools

[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export

[Create a book](#)
[Download as PDF](#)
[Printable version](#)

Languages

[Čeština](#)
[Deutsch](#)
[Français](#)
[한국어](#)
[Nederlands](#)
[日本語](#)
[Polski](#)
[Русский](#)
[Svenska](#)

 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [\ More](#)

Earliest deadline first scheduling

From Wikipedia, the free encyclopedia

Earliest deadline first (EDF) or **least time to go** is a dynamic [scheduling algorithm](#) used in [real-time operating systems](#) to place processes in a [priority queue](#). Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

EDF is an *optimal* scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent *jobs*, each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the **EDF** will schedule this collection of jobs so they all complete by their deadline.

With scheduling periodic processes that have deadlines equal to their periods, **EDF** has a utilization bound of 100%. Thus, the [schedulability test](#) [↗](#) for **EDF** is:

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

where the $\{C_i\}$ are the worst-case computation-times of the n processes and the $\{T_i\}$ are their respective inter-arrival periods (assumed to be equal to the relative deadlines).

That is, EDF can guarantee that all deadlines are met provided that the total [CPU](#) utilization is not more than 100%. Compared to fixed priority scheduling techniques like [rate-monotonic scheduling](#), **EDF** can guarantee all the deadlines in the system at higher loading.

However, when the system is overloaded, the set of processes that will miss deadlines is largely unpredictable (it will be a function of the exact deadlines and time at which the overload occurs.) This is a considerable disadvantage to a real time systems designer. The algorithm is also difficult to implement in [hardware](#) and there is a tricky issue of representing deadlines in different ranges (deadlines can't be more precise than the granularity of the clock used for the scheduling). If a modular arithmetic is used to calculate future deadlines relative to now, the field storing a future relative deadline must accommodate at least the value of the ("duration" {of the longest expected time to completion} * 2) + "now"). Therefore **EDF** is not commonly found in industrial real-time computer systems.

Instead, most real-time computer systems use [fixed priority scheduling](#) (usually [rate-monotonic scheduling](#)). With fixed priorities, it is easy to predict that overload conditions will cause the low-priority processes to miss deadlines, while the highest-priority process will still meet its deadline.

There is a significant body of research dealing with **EDF** scheduling in [real-time computing](#); it is possible to calculate worst case response times of processes in **EDF**, to deal with other types of processes than periodic processes and to use servers to regulate overloads.

Contents [\[hide\]](#)

- 1 [Example](#)
 - 1.1 [Timing diagram](#)
 - 1.2 [Utilization](#)
- 2 [Deadline interchange](#)
- 3 [Heavy traffic analysis for EDF queues with reneging](#)
- 4 [Kernels implementing EDF scheduling](#)
- 5 [See also](#)
- 6 [References](#)

Example [\[edit\]](#)

Consider 3 periodic processes scheduled on a preemptive uniprocessor. The execution times and periods are as shown in the following table:

Process Timing Data

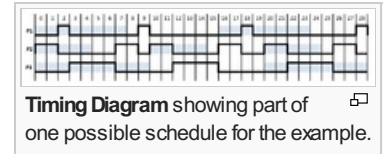
Process	Execution Time	Period

P1	1	8
P2	2	5
P3	4	10

In this example, the units of time may be considered to be schedulable [time slices](#). The deadlines are that each periodic process must complete within its period.

Timing diagram [\[edit\]](#)

In the timing diagram, the columns represent time slices with time increasing to the right, and the processes all start their periods at time slice 0. The timing diagram's alternating blue and white shading indicates each process's periods, with deadlines at the color changes.



The first process scheduled by EDF is P2, because its period is shortest, and therefore it has the earliest deadline. Likewise, when P2 completes, P1 is scheduled, followed by P3.

At time slice 5, both P2 and P3 have the same deadline, needing to complete before time slice 10, so EDF may schedule either one.

Utilization [\[edit\]](#)

The utilization will be:

$$\left(\frac{1}{8} + \frac{2}{5} + \frac{4}{10}\right) = \left(\frac{37}{40}\right) = 0.925 = \mathbf{92.5\%}$$

Since the [least common multiple](#) of the periods is 40, the scheduling pattern can repeat every 40 time slices. But, only 37 of those 40 time slices are used by P1, P2, or P3. Since the utilization, 92.5%, is not greater than 100%, the system is schedulable with EDF.


Deadline interchange [\[edit\]](#)

Main article: [Priority inheritance](#)

Undesirable deadline interchanges may occur with EDF scheduling. When a shared resource is accessed by processes using critical sections within a process (to prevent it from being pre-empted by another process with an earlier deadline waiting for access to the same shared resource), it becomes important for the scheduler to temporarily assign the earliest deadline from amongst the other processes waiting for the resource, to the process while it is within its critical section to prevent the processes with earlier deadlines miss their respective deadline, especially if the process within its critical section has a much longer time to complete and its exit from its critical section and subsequent release of the shared resource may be delayed. Also it may be further delayed by other processes with earlier deadlines which do not share the same resource and thus can preempt it during its critical section. This hazard of deadline interchange within a critical section is analogous to [priority inversion](#) when using [fixed priority pre-emptive scheduling](#).



To speed up the search within a linked list queue, the waiting processes within the list should be sorted according to their deadlines. When a cyclic or new process is given a new deadline, it is then inserted before the first process with a later deadline. This way, the processes with the earliest deadlines are always at the beginning of the list, reducing the time to find them.

Heavy traffic analysis for EDF queues with reneging [\[edit\]](#)

In a heavy-traffic analysis of the behavior of a single-server queue under an [Earliest-Deadline-First \(EDF\) scheduling policy with reneging](#) , the processes have deadlines and are served only until their deadlines elapse. The fraction of "reneged work," defined as the residual work not serviced due to elapsed deadlines, is an important performance measure.

Kernels implementing EDF scheduling [\[edit\]](#)

Although EDF implementations are not common in commercial real-time kernels, here are a few links of open-source and real-time kernels implementing EDF:

- [SHARK](#)  The SHaRK RTOS, implementing various versions of EDF scheduling and resource reservation scheduling algorithms
- [ERIKA Enterprise](#)  ERIKA Enterprise, which provides an implementation of EDF optimized for small

microcontrollers with an API similar to the [OSEK API](#).

- The [Everyman Kernel](#) [↗] The Everyman Kernel implements either EDF or Deadline Monotonic scheduling depending on the user's configuration.
- [MaRTE OS](#) [↗] MaRTE OS acts as a runtime for Ada applications and implements a wide range of scheduling algorithms including EDF.
- The [AQuoSA](#) project constitutes a modification to the Linux kernel enriching the process scheduler with EDF scheduling capabilities. The timing of the scheduling cannot be as precise as in the case of the above hard real-time Operating Systems, yet it is sufficiently precise so as to greatly enhance predictability, and thus fulfill the real-time requirements, of multimedia applications. AQuoSA is one of a few projects that provides real-time scheduling capabilities to unprivileged users on a system in a controlled way, by means of a properly designed access-control model.^[1]
- The [Linux kernel](#) has an earliest deadline first implementation named [SCHED DEADLINE](#) which is available since the release 3.14.
- The [real-time scheduler](#) [↗] developed in the context of the [IRMOS](#) [↗] European Project is a multi-processor real-time scheduler for the Linux kernel, particularly suitable for temporal isolation and provisioning of QoS guarantees to complex multi-threaded software components and also entire [virtual machines](#). For example, when using Linux as host OS and [KVM](#) as hypervisor, IRMOS can be used to provide scheduling guarantees to individual VMs and at the same time [isolate their performance](#) so as to avoid undesired temporal interferences. IRMOS features a combined EDF/FP [hierarchical scheduler](#). At the outer level there is a partitioned EDF scheduler on the available CPUs. However, reservations are multi-CPU, and global FP over multi-processors is used at the inner level in order to schedule the threads (and/or processes) attached to each outer EDF reservation. See also [this article on lwn.net](#) [↗] for a general overview and a short tutorial about the subject.
- Xen has had an EDF scheduler for some time now. The [man page](#) [↗] contains a short description.
- KVM: The scheduler will probably be implemented by the KVM developers at some point in the future^{[[vague](#)]}.
- The [Plan 9 OS from Bell Labs](#) [↗] incorporates EDFI, a "lightweight real-time scheduling protocol that combines EDF with deadline inheritance over shared resources."^[2]
- [RTEMS](#) [↗]: The EDF scheduler will be available in version 4.11. [RTEMS SuperCore](#) [↗]

See also ^{[[edit](#)]}

- [Dynamic priority scheduling](#)

References ^{[[edit](#)]}

- ↑ Cucinotta, Tommaso (April 2008). "Access Control for Adaptive Reservations on Multi-User Systems" [↗]. *14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*.
- ↑ Pierre G. Jansen, Sape J. Mullender, Paul J.M. Havinga, Hans Scholten. [Lightweight EDF Scheduling with Deadline Inheritance](#) [↗]

Categories: [Processor scheduling algorithms](#) | [Real-time computing](#)

This page was last modified on 10 June 2015, at 18:50.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

