

Add new optional time bases v4

Proposal for C23

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

We propose the inclusion of optional macros for time bases that are modeled after ISO 9945's `CLOCK_MONOTONIC`, `CLOCK_PROCESS_CPU_ID`, and `CLOCK_THREAD_CPU_ID`.

History: This is a follow-up of N2402, N2417 and N2460 or parts thereof. WG14 has already voted in favor of the addition of `TIME_MONOTONIC`, `TIME_ACTIVE` and `TIME_THREAD_ACTIVE` so far found no consensus but didn't meet strong opposition, either.

Changes: v4: (1) Improve reference to calendar time and emphasize on monotonicity. (2) Add dealing with `timespec_getres`, since we have that now. (3) Emphasize more on the optional aspect of all the definitions.

1. INTRODUCTION

The interfaces in `time.h` to manipulate time values have grown mostly unattended over the years and present several problems that could be easily avoided with more modern, redesigned interfaces. This paper is concerned with the following problem:

- The standard allows implementations to add more time bases than `TIME_UTC` but gives no guidance in which direction to go with such new base values.
- POSIX already provides normalized semantics for some other time bases than `TIME_UTC`, and it would be good if we could avoid that practices with similar named time base emerge that diverge from these.

1.1. Strategy

C11 and C17 left the addition of new time bases completely to the implementation. Although it is a good principle to leave room for extensions, certain of them already have a connotation in other normative context. In particular, ISO 9945 already provides specifications for four different time bases, two for elapsed time measurement (`CLOCK_REALTIME` and `CLOCK_MONOTONIC`), and two for active processing (CPU) time (`CLOCK_PROCESS_CPUTIME_ID` and `CLOCK_THREAD_CPUTIME_ID`).

C11's `timespec_get` and `TIME_UTC` are modeled after ISO 9945's `clock_gettime` and `CLOCK_REALTIME`, so we propose not handle the latter, and to suppose that the specification for `TIME_UTC` is sufficient.

For the other three, we propose to add optional macros to the standard, such that the names, if defined, bind implementations to a particular semantic. ISO 9945 and ISO 9899 differ slightly in their interfaces and have different terminology, so we propose to have macro names according to C's terminology with a prefix `TIME`:

- `TIME_MONOTONIC` for a time base that is not affected by changes to the time environment, if such a change is possible on the platform. The intent is to provide a measure of time as perceived by the execution platform in its current physical reference system. (This is in contrast to calendar time as measured by `TIME_UTC` which is subject to normative and cultural adjustments.)
- `TIME_ACTIVE` which is the active processing time that is accounted for the whole execution. The intent is to provide a value that is consistent with the return of the `clock` function as specified by the C standard.
- `TIME_THREAD_ACTIVE` which is the same, but accounted on a per thread base.

Since these macros will generally have different values from the ones provided by ISO 9945 (there the constants have the opaque type `clockid_t`) we can impose positive values without invalidating components of ISO 9945.

1.2. Elapsed time

ISO 9945 has two different “clocks” for measurement of elapsed time, `CLOCK_REALTIME` and `CLOCK_MONOTONIC`. They differ eventually in the starting point of the measurement (*epoch* vs. boot time) and, more importantly, concerning their behavior when the system time is set:

- `CLOCK_REALTIME` changes when the clock is set to a new value, *e.g.* if a background time daemon adjusts to a drift indicated by a time servers, or if calendar time is adjusted with a leap second. This is the only clock in ISO 9945 that is mandatory, and as such plays a similar role as `TIME_UTC` for ISO 9899.
- `CLOCK_MONOTONIC` is guaranteed not to be affected by such changes of the system clock and to measure physical time as perceived by the platform.

We propose to model the latter by `TIME_MONOTONIC` and to modify 7.27.1 p2 as follows:

```
...
TIME_UTC
TIME_MONOTONIC
which expands to an integer constants, that designates the UTC calendar
time and monotonic time bases, respectively.
```

And then to add a new paragraph:

3' The definition of macros for time bases other than `TIME_UTC` are optional. If defined, the corresponding time bases are supported by `timespec_get` and `timespec_getres`, and their values are positive.

For `timespec_get` we then add text to the end of 7.27.2.5 p3:

The optional time base `TIME_MONOTONIC` is the same, but the reference point is an implementation-defined time point; different program invocations need not to refer to the same reference points.^{FNT1} For the same program invocation, the results of two calls to `timespec_get` with `TIME_MONOTONIC` such that the first happens before the second shall not be decreasing and the result shall not be affected by implementation-specific functions or external events that change the calendar time of the execution platform. It is implementation-defined if `TIME_MONOTONIC` accounts for time during which the execution platform is suspended.^{FNT2}

With the attached footnotes:

^{FNT1} Commonly, this reference point is the boot time of the execution platform or the start of the execution.

^{FNT2} An execution platform may for example not be able to track physical time that elapsed during suspension in a low power consumption mode.

QUESTION 1. *Shall we adopt `TIME_MONOTONIC` as proposed in N2647 section 1.2?*

1.3. Active processing time

In C17, active processing time during a program invocation can be measured by means of the `clock` function. Unfortunately this functions has several problems, the most sever being that it may overflow without notice after a relatively short execution time, for example after 36 minutes on systems with a signed `clock_t` of width 32 and a `CLOCKS_PER_SEC` value of

1 million. Another disadvantage of **clock** is that there is one legacy C implementation that gets this function fundamentally wrong when compared to the C standard: it accounts for elapsed time instead of active processing time. Repeatedly, this leads to confusion when code is ported from or to conforming platforms. For these reasons we think that **clock** is best deprecated and replaced by an appropriate time base for **timespec_get**. For the time being, we also propose to adapt the wording of for the **clock** function to make its purpose more clear.

ISO 9945 has two such “clocks”, namely **CLOCK_PROCESS_CPUTIME_ID** and **CLOCK_THREAD_CPUTIME_ID**, which we propose to adapt to the needs of the C standard, named **TIME_ACTIVE** and **TIME_THREAD_ACTIVE**. The rationale for this choice of naming is that the C standard neither defines the terms processor nor CPU, and that we want to emphasize that the measured time omits times of inactivity of the executed program. If WG14 prefers, other naming choices would be possible.

Because implementations might need to dynamically distinguish different values for these bases for concurrent program invocations (processes) or threads, the specifications of the values exempts them from being compile time constants (append to 7.27.1 p2):

... **;** and
TIME_ACTIVE
TIME_THREAD_ACTIVE
 which, if defined, expand to integer values, designating overall execution and thread-specific active processing time bases, respectively.

and we add in 7.25.1 p3:

If defined, the value of the optional macro **TIME_ACTIVE** shall be different from the constants **TIME_UTC** and **TIME_MONOTONIC** and shall not change during the same program invocation. The optional macro **TIME_THREAD_ACTIVE** shall not be defined if the implementation does not support threads; its value shall be different from the above, it shall be the same for all expansions of the macro from the same thread, and the value provided for one thread shall not be used by a different thread as base argument of **timespec_get** or **timespec_getres**.

For **timespec_get** itself the text proposal in 7.27.2.5 is then quite simple:

For the optional time bases **TIME_ACTIVE** and **TIME_THREAD_ACTIVE** the result is similar, but the call measures the amount of active processing time associated with the whole program invocation or with the calling thread, respectively.

Calls with **TIME_ACTIVE** could replace calls of **clock**, now that we would also know how to query the resolution of this time base with **timespec_getres**. Therefore we propose to add a recommended practice at the end of 7.27.2.5:

Recommended practice

5 It is recommended that timing results of calls to **timespec_get** with **TIME_ACTIVE**, if defined, and of calls to **clock** are as close to each other as their types and resolutions (obtained with **timespec_getres** and **CLOCKS_PER_SEC**, respectively) allow.

QUESTION 2. Shall we adopt **TIME_ACTIVE** and **TIME_THREAD_ACTIVE** as proposed in N2647 section 1.3 ?

1.4. Adjust the `clock`

Optionally, we also propose to modify the wording for the `clock` function, 7.27.2.1 p3:

The clock function returns the implementation's best approximation ~~to the processor time used by the program~~ of the active processing time associated with the program execution since the beginning of an implementation-defined era related only to the program invocation.

QUESTION 3. *Shall we change the wording for the `clock` function as proposed in N2647 1.4?*

In addition, it could be good to indicate the future direction for these interfaces. Therefore we could add the following to "Future library directions", 7.32.13:

2 The function `clock` is an obsolescent feature; `timespec_get` with time base `TIME_ACTIVE` should be used instead by new code whenever possible. The time base `TIME_ACTIVE` may become mandatory in future versions of this standard.

QUESTION 4. *Shall we add the wording for the obsolescence of `clock` and requirement of `TIME_ACTIVE` as proposed in N2647 1.4 the future library directions?*

Acknowledgments

We thank Rajan Bhakta for feedback and discussions.