Long Idle Tasks:
Coupling wagons to the Blink Midnight Train

rmcilroy@chromium.org

(Public document)

Introduction

The Blink Scheduler enables Chrome to make informed decisions regarding which tasks get executed by the main thread and also provides the opportunity to schedule best-effort work which need only be performed when time allows, without impacting higher priority work. These idle tasks are currently scheduled based on the compositor's `WillBeginFrame` signals such that slack time between frames can be used to perform best-effort idle work. This ensures that we avoid Jank, but limits idle tasks to <10ms of deadline and does not take advantage of inter-frame idle time, where Chrome isn't drawing frames at all.

This document discusses an extension to the idle time mechanism which enables longer idle time tasks when no animation or frame drawing is occurring, without impacting the perceived responsiveness of web pages to user-input. This will provide a mechanism on which to implement the <50ms background tasks described in the Blink Midnight Train manifesto.

Background

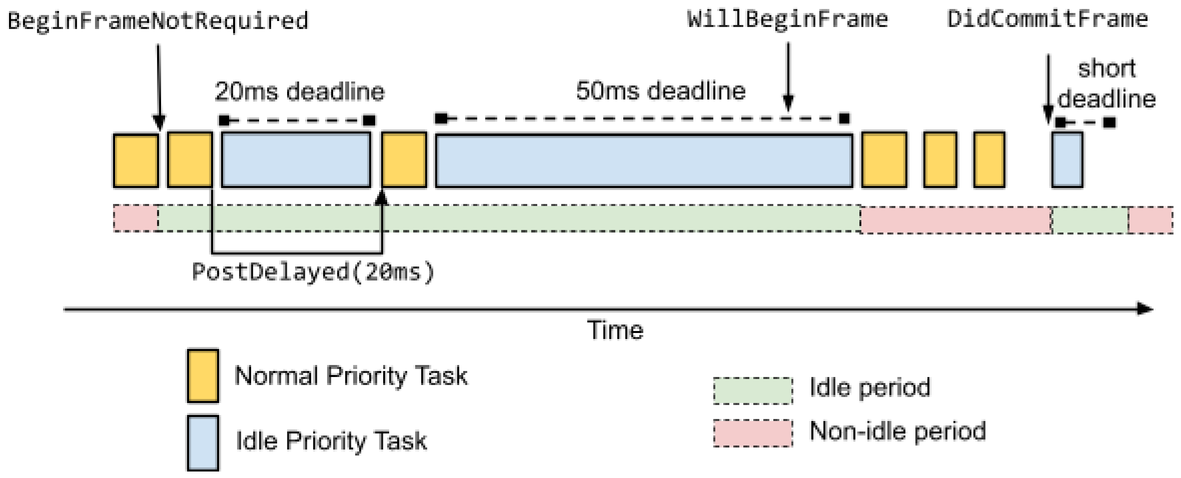
See the Blink Idle Task Scheduling design document for details of how short intra-frame idle tasks are currently scheduled by the Blink Scheduler.

Design

In order to enable longer idle task the Blink scheduler needs to know when the compositor has stopped drawing frames. Currently the compositor sends `WillBeginFrame` notifications to the Blink scheduler when it is beginning a frame draw, and `DidCommitFrameToCompositor` when that frame has been committed. This enables the Blink scheduler to keep track of the intra-frame slack time, however it does not provide enough information for the Blink scheduler to know that no further frames will be drawn during a given period of time.

In order to provide the scheduler with this information we will add a new signal `NoBeginFrameRequired` which the cc scheduler will send to the Blink scheduler when it determines that a new frame will not be required (e.g., when either `BeginFrameNeeded or ShouldSetNeedsBeginFrames returns false due to no more frames being required during SetupNextBeginFrameIfNeeded`). This will cause the Blink scheduler to start an idle period with long idle deadlines. This idle period will last until the cc scheduler sends a new `WillBeginFrame` signal.

During the idle period, idle tasks will only ever run if no other task are pending in either of the other queues (e.g., the `DefaultTaskQueue` or the `CompositorTaskQueue`). The idle tasks will also have a deadline which limits them to the minimum of either 50ms or the earliest delayed task which is pending on the run queues. By limiting the deadline of idle task to a maximum of 50ms, this should allow us to respond to an input event which occurs during this idle period within the 100-150ms lag deadline (assuming that the input event processing code takes no more than 50-100ms itself to execute).



Example long idle-time execution schedule

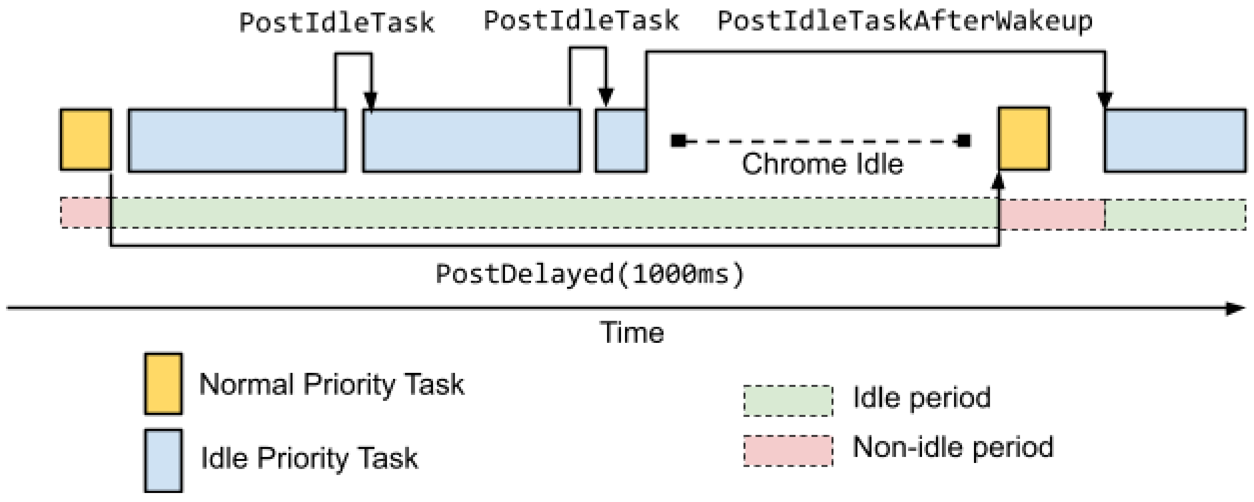
Idle task control

Currently the Blink scheduler limits idle task execution to the intra-frame slack period, and prevents tasks posted from one idle period from executing in the same idle period (the idle queue doesn't get "pumped" until a new idle period starts). This provides a natural 'brake' on idle task execution time - preventing a task can't do any work within the current deadline, but wants to repost itself for a future deadline from creating a "repost storm" where the task repeatedly reposts itself for the remainder of the idle period (thus burning CPU power unnecessarily).

With the long idle time tasks, we don't want to limit components to a single 50ms idle task during the whole long idle time - we want components to be able to use the whole of the idle period to do any necessary background work, as long as they yield back to the scheduler each 50ms to prevent noticeable latency to input events. Therefore, during long idle periods we will reinstate auto-pumping for the idle task queue. [TODO: we should ensure that idle tasks don't reject a 50ms deadline as "too short" otherwise they will never do any idle work and could cause a reposting storm in this situation]

Idle tasks should only repost themselves if they know there is work which they could usefully do in future, however, for some tasks (e.g., V8 idle GC tasks) it is very difficult to determine this (since any JS execution could cause additional GC work, there would need to be check whether a repost is required whenever JavaScript is executed). The V8 GC idle tasks currently sidestep this by simply re-post themselves to be run during the next idle period even if they don't currently have any work to do. This currently works because it enables the idle task to do it's own check for whether it has any background work to do, and the manual pumping of the idle queue ensures that any overhead is limited to one check per frame, however for 'long idle times' there will be no such limit, and even if there was, this approach would cause Chrome to wake-up every 50ms for an unnecessary idle GC task, burning CPU power unnecessarily.

In order to avoid this situation, a new `PostIdleTaskAfterWakeup` API will be added. This will enable idle tasks which have no current work to schedule themselves to be reposted after Chrome has woken up (i.e., after another non-idle task, such as a timer event or an input event). The following diagram shows an example execution:



Example long idle-time execution schedule

Open questions

- Will the cc scheduler be invoked whenever Chrome decides it no longer needs any frames such that the `NoBeginFrameRequired` message can be generated
- Are there situations where a `WillBeginFrame` message won't be sent, but the Compositor is still busy drawing a frame
    - This may apply if there is an active tree currently being worked on
- Should we reenable auto-pumping during long idle periods, or prevent an idle task which reposted itself from being run until a 50ms time period has elapsed (even if there is no other work to do during this time)
- Is there a better approach than `PostIdleTaskAfterWakeup` which solves the same issues

- Would the mechanism to post a task after wakeup be more generally useful - i.e., should we replace `PostIdleTaskAfterWakeup` with a more generic `PostTaskAfterWakeup` which runs as a normal task (and could do the repost of the idle task itself).
- Oilpan might benefit from being able to extend a task. Ross suggests in the comments above that it could do that by periodically polling ShouldYieldForHighPriorityTask. To make this useful, Oilpan could add support for cancelling GCs that are in progress. If a high priority task arrives, we would drop the GC, and zero out the mark bits, restarting GC at a later time. This burns battery, so you wouldn't want to do it too much, but it would enable Oilpan to do optimistic long jank-free GCs when the page was idle, and it would be a lot less work than incremental GC, which is probably still the aim in the long run (post-coming-out-from-under-the-ifdef). One issue is that the jank-measurement infrastructure should not penalize Oilpan for spending time on a task that it *could* have interrupted, had anything come up (is this an issue?).

Related Documents

- Blink Idle Task Scheduling
- Blink Scheduler Refactor
- Cooperative scheduling in JS