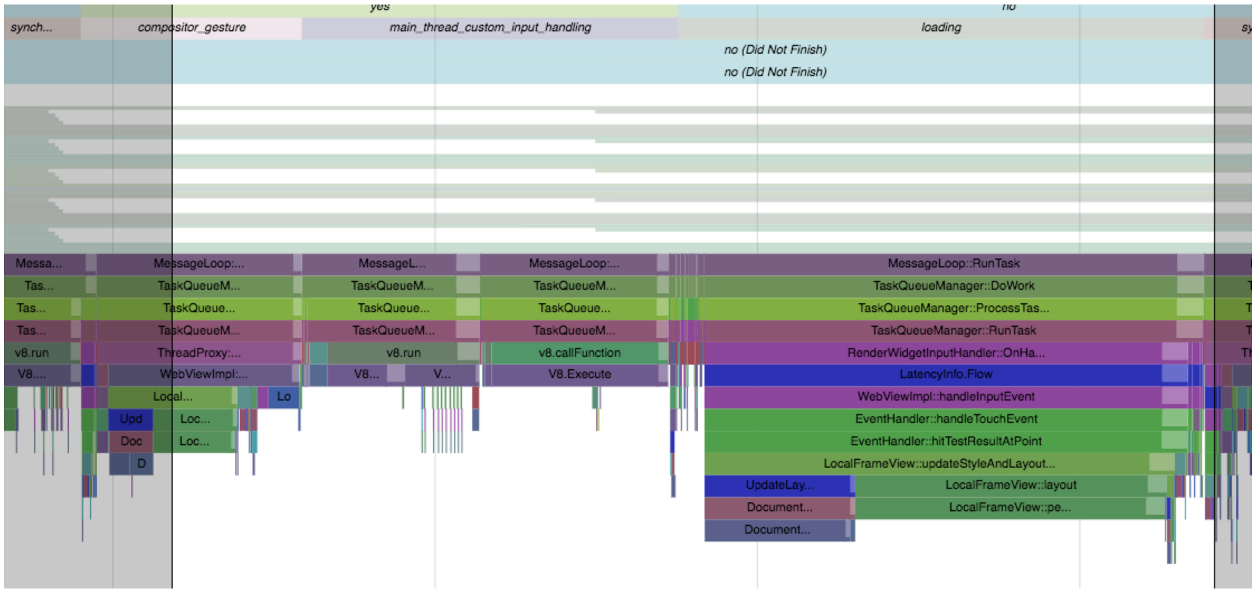
Looking into scrolling lag

altimin@chromium.org, dtapuska@chromium.org
Dec 21, 2017

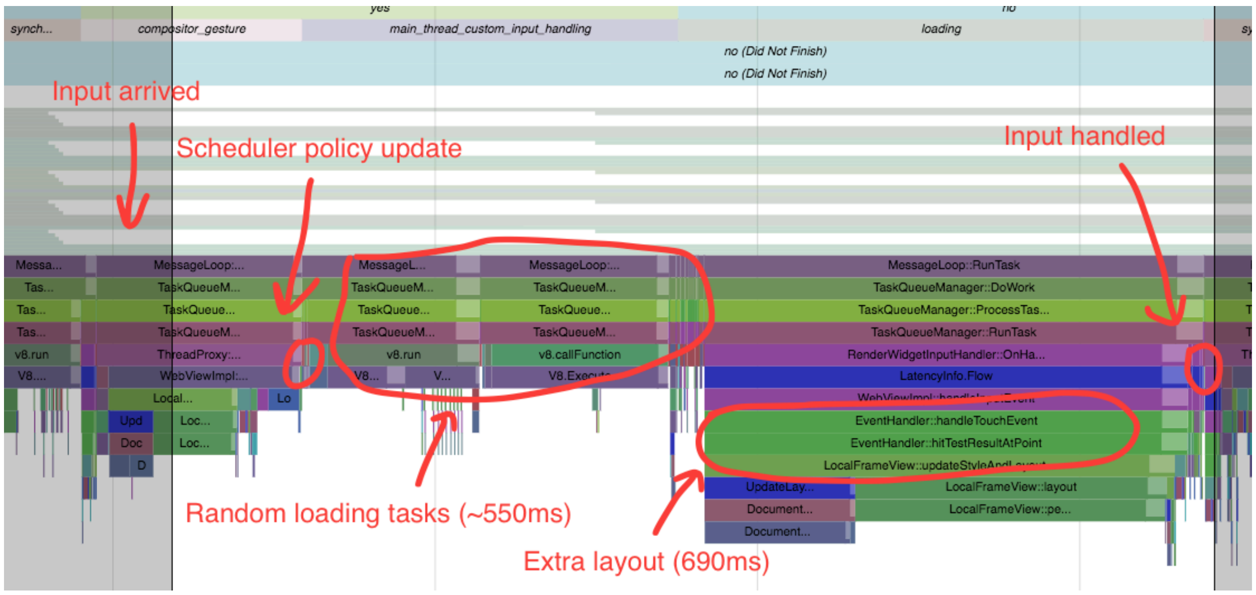


Trace: [guardian.html]

Problem

This trace shows Nexus 5X device rendering [https://theguardian.com] website. It takes 1.6 seconds to respond to a scroll.

What happened


Input arrived
Scheduler policy update
Input handled
Random loading tasks (~550ms)
Extra layout (690ms)
compositor_gesture
main_thread_custom_input_handling
loading
MessageLoop::RunTask
TaskQueueManager::DoWork
TaskQueueManager::ProcessTas...
TaskQueueManager::RunTask
RenderWidgetInputHandler::OnHa...
LatencyInfo.Flow
EventHandler::handleTouchEvent
EventHandler::hitTestResultAtPoint
LocalFrameView::layout
v8.callFunction

Input arrived during an event (BeginMainFrame, which also happened to perform a layout). Immediately after that scheduler updated policy (control queue tasks, `UpdateForInputEventOnCompositorThread` and `UpdatePolicyLocked`). The selected use case was main_thread_custom_input_handling and we failed to prioritise compositor task queue due to compositor tasks being expensive (e.g. last BMF took 80ms). That meant that we started running all tasks of normal priority in order, including loading tasks, which took 550ms. These loading tasks invalidated the layout tree, so when an input task had to run, it took additional 690ms to recompute it.

How can we fix it

RendererScheduler's compositor task queue consists of mainly two things at the moment -- BeginMainFrame related tasks coming from cc/trees/proxy_impl.cc and input-related tasks coming from content/renderer/input/main_thread_event_queue.cc. At the moment scheduler does not distinguish between them, which leads to many unfortunate limitations like not being able to prioritise compositor tasks during main_thread_custom_input_handling due to danger of starving other queues.

It is proposed to split a separate input task queue from compositor task queue and use it in content/renderer/input. This will allow us to always run input events with highest priority.

Q: Will it change any ordering assumptions?
A: It seems that no. The usage of this new task queue will be isolated to MainThreadEventQueue::PostTaskToMainThread. Also existing logic handles all pending events

inside BeginMainFrame task before handling BeginMainFrame itself, meaning that there are no reordering issues.

Estimated amount of work

Low. 1-2 weeks of engineering work for initial implementation and Canary/Dev trial. There existings probability of breakage which will require a further work, but it’s low. This presents us with excellent cost-to-benefit ratio with significant scroll latency improvement at a low cost.

Random notes:

- We need to measure cost of layouts happening during input processing. They can add more delay than the rest of the tasks.
- During a loading task (AddClient from platform/loader/fetch/Resource.cpp) we call ResetForNavigationLocked and fall into loading use case.