Java and C++ task scheduling

Background

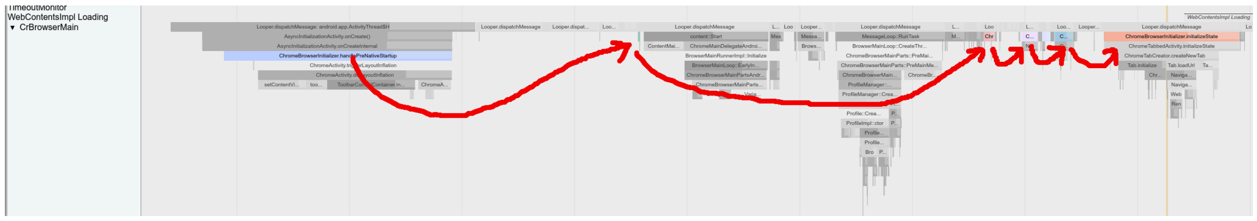
The android UI thread is based around the native android looper. On top of that sits the Java View which has a Handler (used to post to the looper). The Looper is just a simple queue with no prioritization.

Clank code uses helpers to post Java tasks onto the Looper (see ThreadUtils.java) and message_pump_android.cc uses JNI to post C++ tasks onto the UI thread Handler (see SystemMessageHandler.scheduleWork). I.e. Java and C++ tasks are posted onto the same fifo.

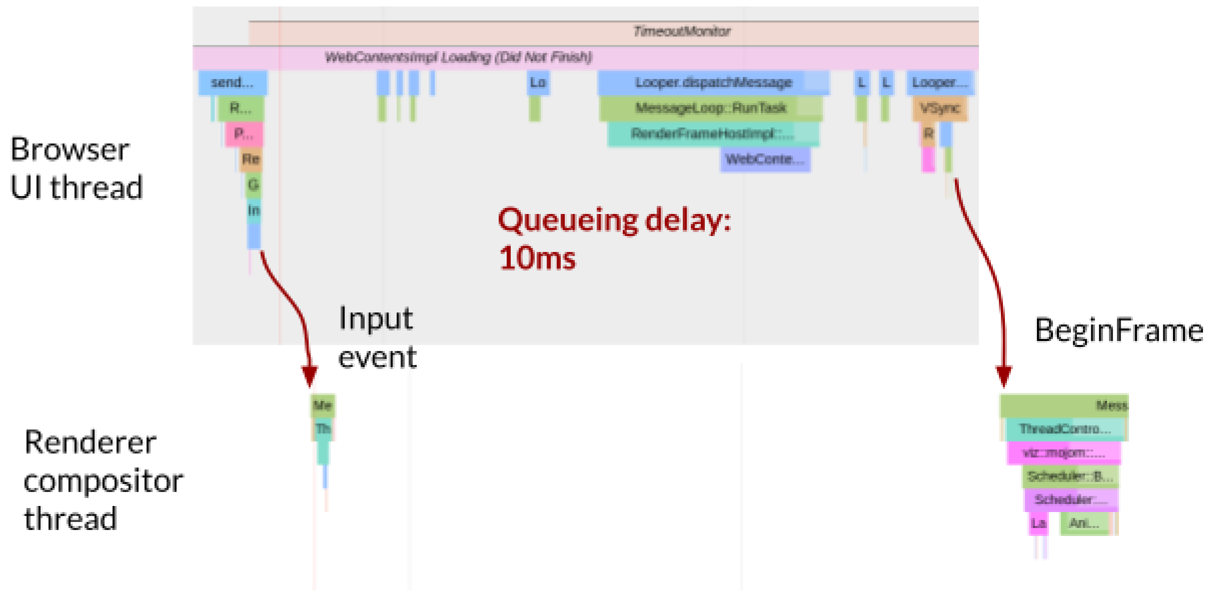
This setup is unfortunate for several reasons:

- There's no prioritization of tasks on the critical path for start up
- There's no prioritization of high priority C++ tasks vs low priority Java tasks or vice versa

This trace shows the critical path for navigation, because there's no prioritization this is longer than it needs to be. NB some of the grey tasks are also necessary for navigation e.g. creating the profile.



The following trace of scrolling the site hs.fi shows an example where lack of prioritization causes a frame drop. Since touch events and the vsync signal (which triggers rendering) are delivered by separate Java tasks, it's possible for C++ tasks to run between them. In this case, navigation related tasks delayed the the vsync task by 10 ms, causing the renderer to miss the frame deadline.



Design

In an ideal world we would hog the UI thread and only yield when there are pending messages or events. We can tell if the Looper is empty but we don't have any way to tell if there's pending input (unless we can somehow get hold of the input file descriptor, see NativeInputEventReceiver::setFdEvents). This means we can only safely run one task at a time before yielding or we risk delaying input handling.

This restriction is very similar to the way the Blink Scheduler, which sits on top of the base MessageLoop and typically runs one task before yielding. We're actively moving the SequenceManager and TaskQueue to base, and are going to use the SequenceManager on the UI thread anyway. So I propose it makes sense to port roughly the same interface (or a subset), and selection logic to Java. We will then refactor clank code to post tasks on appropriate task queues and prioritize them accordingly.

C++ tasks will be posted to a task queue. Before running the java selector, we will use a JNI call to run the C++ selector (possibly simplified because we don't need this to update starvation counters) to determine the priority of the next C++ task. This priority will be applied to the task queue in java which will result in proper prioritization of C++ and Java tasks.

The plumbing of the JNI call through to the SequenceManager needs a little thought. If the SequenceManager was able to talk to the MessagePump directly this would be quite straight forward. For now perhaps the MessageLoop can be passed a repeating callback which it passes to the message pump which message_pump_android.cc will use in the implementation of the new JNI function.