Blink and Task Scheduler integration (PUBLIC)

alexclarke@, altimin@, fdoray@, gab@, robliao@, skyostil@
November 7th, 2017
go/jolly-jumper
Bug: crbug.com/783309
Status: Design in progress
(sign-in with @chromium.org account or request access to comment)

This document describes the merge of the Blink Scheduler with the Task Scheduler (Lucky Luke). The benefit of this unification is that it lets us perform similar dynamic task prioritization on other threads (IO, UI) as we do on the Blink main thread. This can, for example, reduce the latency of input or network event processing. Additionally the Blink Scheduler can virtualize time, which can help make tests more reliable.

Summary

1. Remove Blink dependencies from //third_party/WebKit/Source/platform/scheduler/base.
2. Move .../scheduler/base to //base/task_runner/sequence_manager and add type aliases in Blink to keep things compiling.
3. (parallel) Rewrite Blink-side to remove type aliases.
4. Implement nested run loop support in TaskScheduler and change TaskQueueManager to observe the RunLoop instead of the delegate.
5. Rename TaskQueueManager to SequenceManager
6. Implement a SchedulerLoop.
7. Implement the interfaces specified below in TaskScheduler and SequenceScheduler and change RendererSchedulerImpl (et al) to use the new way for creating a SequenceScheduler.

Directory Structure

Expand **base/task/**

- ➔ **post_task.h** (public interface to TaskScheduler -- moved from base/task_scheduler/post_task.h)
- ➔ **sequence_manager.h** (public interface to SequenceManager -- moved from Blink's TaskQueueManager)
- ➔ **base/task/task_scheduler/*** (remainder of base/task_scheduler/, i.e. impl)
- ➔ **base/task/sequence_manager/*** (remainder of third_party/WebKit/Source/platform/scheduler/base/)

base/task previously only held CancellableTaskTracker.

In the end we'll have CancellableTaskTracker and the public interfaces of the TaskScheduler and SequenceManager in base/task and impls in underlying directories.

Tangent: This would also be a good place to move base/*task_runner.h and *task_runner_handle.h headers to as well.

base/task/task_scheduler/OWNERS <=
  base/task_scheduler/OWNERS
base/task/sequence_manager/OWNERS <=
  third_party/WebKit/Source/platform/scheduler/OWNERS

Work items for the move in 8 steps :
https://docs.google.com/document/d/1bDI7QTvHoB64NINkzkfjro4pTUaBv4buY1gvyQYxAxE/edit

Interfaces

API for SequenceManager to schedule work in TaskScheduler:

```
class WorkerController {
  public:
    // Schedule an immediate call to TakeTask(). Can't be cancelled.
    virtual void ScheduleWork() = 0;

    // Schedule delayed call to TakeTask(). Can only be called on
    // the main sequence.
    virtual void ScheduleDelayedWork(base::TimeTicks time) {
      // SingleThread: just schedule thread wake-up.
      // Sequenced: thread hop to service thread.
    }

    virtual void CancelDelayedWork() = 0;

    // Can only be called on the main sequence. V2 feature?
    virtual void SetPriority(...) = 0;
  }
};
```

The SequenceManager itself implements the following interface for TaskScheduler to call into:

```
class SequencedTaskSource {
 public:
  // Returns a task if a task should be run.
  virtual optional<Task> TakeTask() = 0;
```

```
  // Must be called if TakeTask() returns a task. Returns true if the caller
  // should call TakeTask() at some future time.
  virtual bool DidRunTask() = 0;
};
```

Example Sketch Code:

Main Thread Case:

```
auto task = sequence->TakeTask();
while (task) {
  ExecuteTask(task)
  sequence->DidRunTask();

  task = sequence->TakeTask();
}
```

Shared Worker Case:

```
for (;;) {
  auto sequence = priority_queue->PopSequence();
  auto task = sequence->TakeTask();
  if (!task)
    continue;

  ExecuteTask(task)
  if (sequence->DidRunTask()) {
    priority_queue->Push(sequence);
  }
}
```

API for creating a SequenceManager:

```
// Creation for a thread that’s not created by TaskScheduler (e.g., main thread).
std::unique_ptr<SequenceManager>
base::sequence_manager::TakeOverCurrentThread(
    base::TaskTraits traits) {
  // SchedulerLoop initially wraps a MessageLoop.
  auto loop = SchedulerLoop::GetInstance();
  auto controller = loop->GetController();
  return std::make_unique<SequenceManager>(controller);
}

// Creation for a new thread (e.g., WebWorker thread, IO thread).
void base::sequence_manager::CreateManagedThread(
    MessageLoop::Type type, base::TaskTraits traits,
    base::Callback<std::unique_ptr<SequenceManager>> init_on_thread) {
  auto task_scheduler =
      reinterpret_cast<TaskSchedulerImpl*>(TaskScheduler::GetInstance());
  task_scheduler->CreateSchedulerWorker(type, [](WorkerController* controller) {
```

```
    init_on_thread(std::make_unique<SequenceManager>(controller));
  });
}

// Creation for a non-thread-affine SequenceManager.
std::unique_ptr<SequenceManager>
base::sequence_manager::CreateManagedSequence(
    base::TaskTraits traits) {
  // ... V2 ...
}
```

The Blink Scheduler will be initialized as follows:

```
std::unique_ptr<RendererScheduler> RendererScheduler::Create() {
  std::unique_ptr<SequenceManager> sequence_manager = TakeOverCurrentThread();
  return std::make_unique<RendererSchedulerImpl>(std::move(sequence_manager));
}
```

Changes for workerpoolification

The following changes and constraints will become relevant when we decide to let SequenceManager run in a worker pool. This isn't necessary for the first phase since SequenceManager owns the entire thread it manages.

1. Change all TLS/thread checks in SequenceManager to sequence checks.
2. Query priority from SequenceManager or add a SetPriority() call on the WorkerController and have SequenceScheduler call it based on the priority of the frontmost task.
3. ScheduleWork calling constraints -- allowing multiple scheduled work calls
4. Delays always have a thread hop
5. Delayed task is always a thread hop