```
#ifndef SRC_NODE_PLATFORM_H_
#define SRC_NODE_PLATFORM_H_

#include <queue>
#include <vector>

#include "libplatform/libplatform.h"
#include "node_mutex.h"
#include "uv.h"

namespace node {

template <class T>
class TaskQueue {
 public:
  TaskQueue();
  ~TaskQueue() {}

  void Push(T* task);
  T* Pop();
  T* BlockingPop();
  void NotifyOfCompletion();
  void BlockingDrain();
  void Stop();

 private:
  Mutex lock_;
  ConditionVariable tasks_available_;
  ConditionVariable tasks_drained_;
  int outstanding_tasks_;
  bool stopped_;
  std::queue<T*> task_queue_;
};

class NodePlatform : public v8::Platform {
 public:
  NodePlatform(int thread_pool_size, uv_loop_t* loop,
               v8::TracingController* tracing_controller);
  virtual ~NodePlatform() {}

  void DrainBackgroundTasks();
  // Returns true iff work was dispatched or executed.
  bool FlushForegroundTasksInternal();
```

```
  void Shutdown();

  // v8::Platform implementation.
  size_t NumberOfAvailableBackgroundThreads() override;
  void CallOnBackgroundThread(v8::Task* task,
                              ExpectedRuntime expected_runtime) override;
  void CallOnForegroundThread(v8::Isolate* isolate, v8::Task* task) override;
  void CallDelayedOnForegroundThread(v8::Isolate* isolate, v8::Task* task,
                                     double delay_in_seconds) override;
  bool IdleTasksEnabled(v8::Isolate* isolate) override;
  double MonotonicallyIncreasingTime() override;
  v8::TracingController* GetTracingController() override;

 private:
  uv_loop_t* const loop_;
  uv_async_t flush_tasks_;
  TaskQueue<v8::Task> foreground_tasks_;
  TaskQueue<std::pair<v8::Task*, double>> foreground_delayed_tasks_;
  TaskQueue<v8::Task> background_tasks_;
  std::vector<std::unique_ptr<uv_thread_t>> threads_;

  std::unique_ptr<v8::TracingController> tracing_controller_;
};

}  // namespace node

#endif  // SRC_NODE_PLATFORM_H_
```