Document number: *PnnnnR0*
Date: 2024-*05-09*
Audience: Library Evolution Working Group
Reply-to: *“Anshul Mittal <anshulmttl@gmail.com>;*

## I. Table of Contents

## II. Introduction

*I am sending this document to add new standard to C++. When a single thread tries locking on the same mutex std::mutex it creates a deadlock.*

*E.g.*

*#include <mutex>*

*mutex mut;*

*mut lock;*

*mut lock; // This creates a deadlock on same thread which is not required.*

*My proposal is to check for thread ID while locking on a mutex using std::this_thread::get_id() or other means.*

## III. Motivation and Scope

In My project there is a situation where same thread locked on mutex creating deadlock. So i have written my own mutex to handle this problem. It should be included in C++ standard. It will support novice programmers. It is not used till now, it is my own technology.

The thread ID can be checked as follows.

```
struct Mutex1
{
	public:
		Mutex1();
		~Mutex1();

		void lock();
		void unlock();

		//bool SetThreadIDBeforeLocking(uint64_t thread_id);

	private:
		std::mutex m_Mutex;
		uint64_t m_ThreadID;
		bool m_LockMutex;
		bool m_MutexIsLocked;
};
```

```
Mutex1::Mutex1()
{
   m_LockMutex = true;
   m_MutexIsLocked = false;
   m_ThreadID = 0;
}

Mutex1::~Mutex1()
{
   //m_LockMutex = true;
   m_MutexIsLocked = false;
}

void Mutex1::lock()
{
   uint64_t thread_id = std::hash<std::thread::id>{}(std::this_thread::get_id());
   if(m_ThreadID == thread_id)
   {
      // Locking on same thread. Do not allow.
   }
   else
   {
      m_ThreadID = thread_id;
      m_MutexIsLocked = true;
      m_Mutex.lock();
   }
}

void Mutex1::unlock()
{
   if(m_MutexIsLocked)
   {
      m_Mutex.unlock();
      m_MutexIsLocked = false;
      //m_LockMutex = true;
      m_ThreadID = 0;
   }

   m_MutexIsLocked = false;
}
```

**IV. Impact On the Standard**

It depends on std::mutex, std::thread (std::this_thread::get_id()). It requires changes to standard components. It can be implemented using current C++ compilers and libraries.

**V. Design Decisions**

In my project there is lot of code and keeping track of mutexes is difficult. So i created standard mutex to handle the situation of deadlock due to single thread. I wrote my own standard mutex. A mutex should acquire lock on same thread twice.

**VI. Technical Specifications**

*This is the code for standard mutex*

```
struct Mutex1
{
	public:
		Mutex1();
		~Mutex1();

		void lock();
		void unlock();

		//bool SetThreadIDBeforeLocking(uint64_t thread_id);

	private:
		std::mutex m_Mutex;
		uint64_t m_ThreadID;
		bool m_LockMutex;
		bool m_MutexIsLocked;
};

Mutex1::Mutex1()
{
   m_LockMutex = true;
   m_MutexIsLocked = false;
   m_ThreadID = 0;
}

Mutex1::~Mutex1()
{
   //m_LockMutex = true;
   m_MutexIsLocked = false;
}

void Mutex1::lock()
{
   uint64_t thread_id = std::hash<std::thread::id>{}(std::this_thread::get_id());
   if(m_ThreadID == thread_id)
   {
      // Locking on same thread. Do not allow.
   }
```

```
    else
    {
        m_ThreadID = thread_id;
        m_MutexIsLocked = true;
        m_Mutex.lock();
    }
}

void Mutex1::unlock()
{
    if(m_MutexIsLocked)
    {
        m_Mutex.unlock();
        m_MutexIsLocked = false;
        //m_LockMutex = true;
        m_ThreadID = 0;
    }

    m_MutexIsLocked = false;
}
```

## VII. Acknowledgements

## VIII. References