Priority Inversion

```
// Shared state and mutex to synchronise access to it
std::vector<float> vec;
std::mutex mutex;


// Thread 1 - High priority
{
    std::scoped_lock<std::mutex> lock (mutex);      // Lock access to vec
    std::for_each (vec.begin(), vec.end(),          // Process vec
                   [] (auto& f) { f *= 0.1f; });
}

// Thread 2 - Low priority
{
    std::scoped_lock<std::mutex> lock (mutex);      // Lock access to vec
    vec.resize (500'000);                           // Perform expensive operation
}                                                   // Could be de-scheduled
```

```
template <class _Tp>
_LIBCPP_AVAILABILITY_ATOMIC_SHARED_PTR
void
atomic_store(shared_ptr<_Tp>* __p, shared_ptr<_Tp> __r)
{
    __sp_mut& __m = __get_sp_mut(__p);
    __m.lock();
    __p->swap(__r);
    __m.unlock();
}
```