Лекция 32:
C++ thread library

Цели разработки

- Кросс-платформенность: возможность реализации на разных ОС: POSIX, Windows
- Сокрытие деталей реализации (например, манипуляций с указателями в POSIX Thread)
- Следование стилю и духу C++ (RAII, move semantics)
- Поддержка асинхронных операций

std::thread

- Абстракция параллельного потока выполнения
- Нить создается при вызове конструктора
- В конструкторе указывается вызываемая функция и ее аргументы
- До вызова деструктора должен быть либо join, либо detach
- Операция копирования и присваивания с копированием не поддерживается
- Инкапсулирует платформенный handle (например, pthread_t)

Возврат значения из нити

- Возвращаемое значение функции в std::thread игнорируется!
- Нужен механизм для передачи возвращаемого значения нити обратно ожидающей нити
- Нужен механизм для передачи информации об исключении при выполнении нити

Promise-future

- Пара promise-future образует “канал” для передачи данных от нити к другой нити
- Promise - “записывающая” сторона
- Future - “читающая сторона”
- Значение может не передаваться: promise<void>, тогда это просто барьер

std::promise<T>

- Копирование запрещено
- get_future – получить future для ожидания/чтения значения
- set_value – установить значение
- set_exception – установить исключение

std::future<T>

- Хранит результат выполнения нити или исключение, когда что-либо из них будет доступно
- get() - ждать когда значение будет готово (std::promise<T>::set_value) и вернуть его
- wait() - ждать
- Если был выполнен set_exception, то и get/wait выбрасывают исключение

std::packaged_task

- Обертка над std::promise и функцией для проброса возвращаемого значения и исключения
- get_future() - получить std::future для ожидания
- Нельзя копировать

std::async

- Асинхронное выполнение
  - Может быть создается нить
  - А может используется готовая нить (thread pool)
- Режимы выполнения:
  - launch::async
  - launch::deferred - “ленивый” запуск
- Возвращается future для получения результата

Мьютексы

- mutex
- recursive_mutex
- timed_mutex
- recursive_timed_mutex
- Recursive – могут повторно блокироваться той же самой нитью
- Timed – поддерживаются тайм-ауты при ожидании
- Нельзя копировать
- Методы: lock – захватить, unlock - освободить

shared_mutex

- Эксклюзивный режим захвата (lock / unlock)
- Разделяемый режим захвата (lock_shared / unlock_shared)
- Задача “читатели-писатели”

RAII для мьютексов

- lock_guard<M>
  - Захват одного мьютекса в конструкторе
  - Освобождение в деструкторе
- scoped_lock<M...>
  - Захват нескольких мьютексов в конструкторе
  - Освобождение в обратном порядке в деструкторе

Smart pointers for mutex

- std::unique_lock – аналог unique_ptr для mutex
- std::shared_lock – аналог unique_ptr для shared_mutex

std::condition_variable

- Требует дополнительного mutex: std::unique_lock
- notify_one – уведомить одного
- notify_all – уведомить всех
- wait – позволяет задавать предикат окончания ожидания