Доработки библиотеки hpx

> Вырастает из темы: *Доработки hpx*

p0233 – 2017-10-15 HP

p0566 – 2018-05-06 Proposed wording HP and RCU

p1121 – 2019-01-20 Proposed wording and interface for HP

folly – Реализация, приближенная к p0566. Использует части библиотеки folly(SingletonThreadLocal, SingletonManager, folly:Executor).

libcds – Другая реализация HP.

folly умеет использовать не только thread_local, libcds использует только thread_local storage.

В p0233 написано “Due to the performance advantages of using TLS, the library implementation should allow the programmer to choose implementation paths that benefit from TLS when suitable, and avoid TLS when incompatible with the use case.”

libcds

```
cds::Initialize(); once
cds::gc::HP hpGC; once
cds::threading::Manager::attachThread(); every thread
cds::threading::Manager::detachThread(); every thread
```

http://libcds.sourceforge.net/doc/cds-api/index.html#cds_how_to_use

Не умеет в разные HP_domain, умеет только в собственный thread_local tls, не особо гибкий. Хотя умеет в

```
set_memory_allocator(
    void* ( *alloc_func )( size_t size ),   ///< \p malloc() function
    void( *free_func )( void * p )  )
```

class HP – главный класс, Before use any HP-related class you must initialize \p %HP by contructing \p %cds::gc::HP object in beginning of your \p main().

class Guard – A guard is a hazard pointer. Additionally, the Guard class manages allocation and deallocation of the hazard pointer.

```
Guard::protect(atomics::atomic<T> const& toGuard)
Guard::protect(atomics::atomic<T> const& toGuard, Func f)
```

Приводит T* к void* и работает с этими указателями.

```
template <class Disposer, typename T>
static void retire( T * p )
```

Disposer это шаблонный параметр, и на самом деле тип, а не объект. В стандарте должен быть объектом, “Registers the expression reclaim(static_cast<T*>(this)) to be evaluated asynchronously”

if на linux и membarrier

folly

Аналог Guard, protect == get_protected

```
hazptr_holder: Class that owns and manages a hazard pointer.
T* hazptr_holder::get_protected(const Atom<T*>& src) noexcept;
```

В folly Используется T*, в libcds: T

Все T обязаны наследоваться от hazptr_obj_base<T>

```
template <
    typename T,
    template <typename> class Atom = std::atomic,
    typename D = std::default_delete<T>>
class hazptr_obj_base {
    void retire( D deleter = {}, hazptr_domain<Atom>& domain =
default_hazptr_domain<Atom>());
}
```

hazptr_domain в folly есть, но он не умеет в разные аллокаторы, хотя по стандарту должен.

Дополнительные вещи, не из стандарта: hazptr_array<N> для N hazptr-ов сразу, быстрее. hazptr_local<N> немного быстрее, но обязывает иметь ровно 1 активный hazptr_* на поток

p1121 (последний)

```
header <hazard_pointer>
```

```
// ?.?, Class hazard_pointer_domain:
class hazard_pointer_domain;

// ?.?, Default hazard_pointer_domain:
hazard_pointer_domain& hazard_pointer_default_domain() noexcept;

// ?.?, Clean up
void hazard_pointer_clean_up(    hazard_pointer_domain& domain =
```

```
hazard_pointer_default_domain());

// ?.?, Class template hazard_pointer_obj_base:
template <typename T, typename D = default_delete<T>>  class
hazard_pointer_obj_base;4

// ?.?, Class hazard_pointer
class hazard_pointer;

// ?.?, Construct non-empty hazard_pointer
hazard_pointer make_hazard_pointer(    hazard_pointer_domain& domain =
hazard_pointer_default_domain());

// ?.?, Hazard pointer swap
void swap(hazard_pointer&, hazard_pointer&) noexcept;
```

```
class hazard_pointer_domain {
        public:
        // ?.?.? constructor:
        explicit hazard_pointer_domain(
std::pmr::polymorphic_allocator<byte> poly_alloc = {});
      ...
      }
```