

Грязные C++ трюки

из `userver` и `Boost`

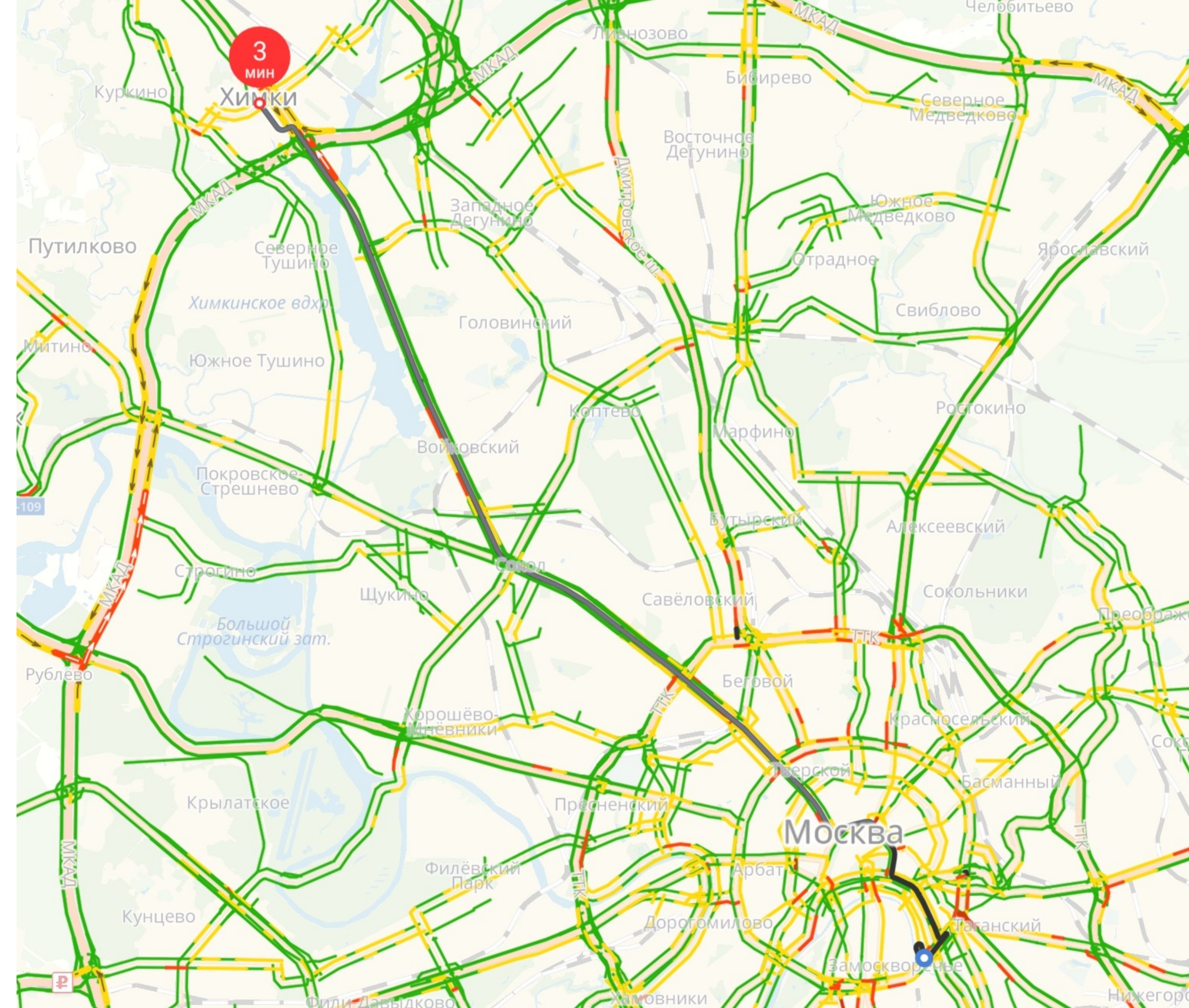
Полухин Антон

Antony Polukhin

Яндекс 

Содержание

- Дополняем все исключения
- Ускоряем исключения
- `reinterpret_cast<void*>(0x42)`
- Получаем имена полей без разметки макросами



● Boost

● userver



ЭКОНОМ
4₽



КОМФОРТ
8₽



КОМФОРТ+
9₽



БИЗНЕС
34₽



МИНИВЭН
15₽



ДЕТСКИЙ
2₽

Комментарий, пожелания

Способ оплаты
Команда Яндекс.Такси



Что делает `throw`?




Что делает `throw`?

Что делает throw?

```
void test(int i) {  
    throw i;  
}
```


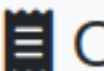

x86-64 clang (trunk) (Editor #1) ✎ ✕

x86-64 clang (trunk) ▾   -O2 ▾



A ▾ ⚙ ▾ ▾ ▾   + ▾ 




```

1 test(int): # @test(int)
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception@PLT
6   mov dword ptr [rax], ebx
7   mov rsi, qword ptr [rip + typeinfo fo
8   mov rdi, rax
9   xor edx, edx
10  call __cxa_throw@PLT
  
```

  Output (0/0) x86-64 clang (trunk) **i** - 414ms (8177B) ~166 lines filtered 




x86-64 gcc (trunk) (Editor #1) ✎ ✕

x86-64 gcc (trunk) ▾   -O2 ▾





A ▾ ⚙ ▾ ▾ ▾   + ▾ 

```

1 test(int):
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception
6   xor edx, edx
7   mov esi, OFFSET FLAT:typeinfo fo
8   mov DWORD PTR [rax], ebx
9   mov rdi, rax
10  call __cxa_throw
  
```

  Output (0/0) x86-64 gcc (trunk) **i** - 663ms (5899B) ~380 lines filtered 


C++ source #1 ✎

A ▾  Save/Load + Add new... ▾  Vim  CppInsights  Quick-bench

```

1 void test(int i) { throw i; }
  
```

Compiler License

 C++ ▾

x86-64 clang (trunk) (Editor #1) ✎ ✕

x86-64 clang (trunk) ▾ -O2 ▾

A ▾ ▾ ▾ + ▾ ▾

```

1 test(int): # @test(int)
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception@PLT
6   mov dword ptr [rax], ebx
7   mov rsi, qword ptr [rip + typeinfo fo
8   mov rdi, rax
9   xor edx, edx
10  call __cxa_throw@PLT
  
```

Output (0/0) x86-64 clang (trunk) - 414ms (8177B) ~166 lines filtered

x86-64 gcc (trunk) (Editor #1) ✎ ✕

x86-64 gcc (trunk) ▾ -O2 ▾

A ▾ ▾ ▾ + ▾ ▾

```

1 test(int):
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception
6   xor edx, edx
7   mov esi, OFFSET FLAT:typeinfo fo
8   mov DWORD PTR [rax], ebx
9   mov rdi, rax
10  call __cxa_throw
  
```

Output (0/0) x86-64 gcc (trunk) - 663ms (5899B) ~380 lines filtered

Compiler License

C++ source #1 ✎

A ▾ Save/Load + Add new... ▾ Vim CppInsights Quick-bench

C++ ▾

```

1 void test(int i) { throw i; }
  
```

x86-64 clang (trunk) (Editor #1) ✎ ✕

x86-64 clang (trunk) ▾ -O2 ▾

A ▾ ▾ ▾ + ▾ ▾

```

1 test(int): # @test(int)
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception@PLT
6   mov dword ptr [rax], ebx
7   mov rsi, qword ptr [rip + typeinfo fo
8   mov rdi, rax
9   xor edx, edx
10  call __cxa_throw@PLT
  
```

Output (0/0) x86-64 clang (trunk) - 414ms (8177B) ~166 lines filtered

x86-64 gcc (trunk) (Editor #1) ✎ ✕

x86-64 gcc (trunk) ▾ -O2 ▾

A ▾ ▾ ▾ + ▾ ▾

```

1 test(int):
2   push rbx
3   mov ebx, edi
4   mov edi, 4
5   call __cxa_allocate_exception
6   xor edx, edx
7   mov esi, OFFSET FLAT:typeinfo fo
8   mov DWORD PTR [rax], ebx
9   mov rdi, rax
10  call __cxa_throw
  
```

Output (0/0) x86-64 gcc (trunk) - 663ms (5899B) ~380 lines filtered

Compiler License

C++ source #1 ✎

A ▾ Save/Load + Add new... ▾ Vim CppInsights Quick-bench

C++ ▾

```

1 void test(int i) { throw i; }
  
```


А что такое `__cxa_allocate_exception` ?

Code

Blame

Raw



```
77  /* Everything from libstdc++ is weak, to avoid requiring that library
78     to be linked into plain C applications using libitm.so.  */
79
80  #define WEAK __attribute__((weak))
81
82  extern "C" {
83
84  ✓ struct __cxa_eh_globals
85  {
86      void *      caughtExceptions;
87      unsigned int uncaughtExceptions;
88  };
89
90  extern void * __cxa_allocate_exception (size_t) _ITM_NOTHROW WEAK;
91  extern void __cxa_free_exception (void *) _ITM_NOTHROW WEAK;
92  extern void __cxa_throw (void *, void *, void (*) (void *)) WEAK;
93  extern void * cxa_begin_catch (void *) ITM_NOTHROW WEAK;
```

Code

Blame

Raw



```
77  /* Everything from libstdc++ is weak, to avoid requiring that library
78     to be linked into plain C applications using libitm.so.  */
79
80  #define WEAK __attribute__((weak))
81
82  extern "C" {
83
84  struct __cxa_eh_globals
85  {
86      void *      caughtExceptions;
87      unsigned int  uncaughtExceptions;
88  };
89
90  extern void *__cxa_allocate_exception (size_t) _ITM_NOTHROW WEAK;
91  extern void __cxa_free_exception (void *) _ITM_NOTHROW WEAK;
92  extern void __cxa_throw (void *, void *, void (*) (void *)) WEAK;
93  extern void * __cxa_begin_catch (void *) _ITM_NOTHROW WEAK;
```

```
392
393 extern "C" void *
394 __cxxabiv1::__cxa_allocate_exception(std::size_t thrown_size) noexcept
395 {
396     thrown_size += sizeof (__cxa_refcounted_exception);
397
398     void *ret = malloc (thrown_size);
399
400     #if USE_POOL
401         if (!ret)
402             ret = emergency_pool.allocate (thrown_size);
403     #endif
404
405     if (!ret)
406         std::terminate ();
407
408     memset (ret, 0, sizeof (__cxa_refcounted_exception));
409
410     return (void *)((char *)ret + sizeof (__cxa_refcounted_exception));
411 }
412
413
```

```
392
393 extern "C" void *
394 __cxxabiv1::__cxa_allocate_exception(std::size_t thrown_size) noexcept
395 {
396     thrown_size += sizeof (__cxa_refcounted_exception);
397
398     void *ret = malloc (thrown_size);
399
400     #if USE_POOL
401         if (!ret)
402             ret = emergency_pool.allocate (thrown_size);
403     #endif
404
405     if (!ret)
406         std::terminate ();
407
408     memset (ret, 0, sizeof (__cxa_refcounted_exception));
409
410     return (void *)((char *)ret + sizeof (__cxa_refcounted_exception));
411 }
412
413
```

```
392
393 extern "C" void *
394 __cxxabiv1::__cxa_allocate_exception(std::size_t thrown_size) noexcept
395 {
396     thrown_size += sizeof (__cxa_refcounted_exception);
397
398     void *ret = malloc (thrown_size);
399
400     #if USE_POOL
401         if (!ret)
402             ret = emergency_pool.allocate (thrown_size);
403     #endif
404
405     if (!ret)
406         std::terminate ();
407
408     memset (ret, 0, sizeof (__cxa_refcounted_exception));
409
410     return (void *)((char *)ret + sizeof (__cxa_refcounted_exception));
411 }
412
413
```

```
392
393 extern "C" void *
394 __cxxabiv1::__cxa_allocate_exception(std::size_t thrown_size) noexcept
395 {
396     thrown_size += sizeof (__cxa_refcounted_exception);
397
398     void *ret = malloc (thrown_size);
399
400     #if USE_POOL
401         if (!ret)
402             ret = emergency_pool.allocate (thrown_size);
403     #endif
404
405     if (!ret)
406         std::terminate ();
407
408     memset (ret, 0, sizeof (__cxa_refcounted_exception));
409
410     return (void *)((char *)ret + sizeof (__cxa_refcounted_exception));
411 }
412
413
```

```
392
393 extern "C" void *
394 __cxxabiv1::__cxa_allocate_exception(std::size_t thrown_size) noexcept
395 {
396     thrown_size += sizeof (__cxa_refcounted_exception);
397
398     void *ret = malloc (thrown_size);
399
400     #if USE_POOL
401         if (!ret)
402             ret = emergency_pool.allocate (thrown_size);
403     #endif
404
405     if (!ret)
406         std::terminate ();
407
408     memset (ret, 0, sizeof (__cxa_refcounted_exception));
409
410     return (void *)((char *)ret + sizeof (__cxa_refcounted_exception));
411 }
412
413
```


Итак, что мы имеем?

Итак, что мы имеем?

`throw что-то;`

Итак, что мы имеем?

throw что-то;

- Аллоцирует место под
 - Заголовок исключения
 - Тело исключения

Итак, что мы имеем?

throw что-то;

- Аллоцирует место под
 - Заголовок исключения
 - Тело исключения
- Зануляет заголовок исключения

Итак, что мы имеем?

throw что-то;

- Аллоцирует место под
 - Заголовок исключения
 - Тело исключения
- Зануляет заголовок исключения
- Возвращает указатель на место для тела исключения

Итак, что мы имеем?

throw что-то;

- Аллоцирует место под
 - Заголовок исключения
 - Тело исключения
- Зануляет заголовок исключения
- Возвращает указатель на место для тела исключения
- По указателю компилятор размещает исключение

Итак, что мы имеем?

`throw` что-то;

- Аллоцирует место под
 - Заголовок исключения
 - Тело исключения
- Зануляет заголовок исключения
- Возвращает указатель на место для тела исключения
- По указателю компилятор размещает исключение
- Вызывается `__сха_throw`

Итак, что мы имеем?

throw что-то;

- **Аллоцирует место под**
 - **Заголовок исключения**
 - **Тело исключения**
- **Зануляет заголовок исключения**
- **Возвращает указатель на место для тела исключения**
- По указателю компилятор размещает исключение
- Вызывается `__сха_throw`

**А что интересного можно с
этим сообразить?**

Сделаем подмешивание `stacktrace` к каждому `throw`!

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__cxa_allocate_exception`

```
extern "C" BOOST_SYMBOL_EXPORT
```

```
void* __cxa_allocate_exception(size_t thrown_size) throw()
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`

Сделаем подмешивание stacktrace к каждому throw!

– Создаём свою функцию `__cxa_allocate_exception`

– В ней:

• Получаем изначальный `__cxa_allocate_exception`

```
static const auto orig_allocate_exception = []() {  
    void* ptr = dlsym(RTLD_NEXT, "__cxa_allocate_exception");  
    BOOST_ASSERT_MSG(ptr, "Failed to find '__cxa_...');  
    return reinterpret_cast<void*(*)(std::size_t)>(ptr);  
}();
```


Сделаем подмешивание stacktrace к каждому throw!

– Создаём свою функцию `__cxa_allocate_exception`

– В ней:

• Получаем изначальный `__cxa_allocate_exception`

```
static const auto orig_allocate_exception = []() {  
    void* ptr = dlsym(RTLD_NEXT, "__cxa_allocate_exception");  
    BOOST_ASSERT_MSG(ptr, "Failed to find '__cxa_...');  
    return reinterpret_cast<void*(*)(std::size_t)>(ptr);  
}();
```

Сделаем подмешивание stacktrace к каждому throw!

– Создаём свою функцию `__cxa_allocate_exception`

– В ней:

• Получаем изначальный `__cxa_allocate_exception`

```
static const auto orig_allocate_exception = []() {  
    void* ptr = dlsym(RTLD_NEXT, "__cxa_allocate_exception");  
    BOOST_ASSERT_MSG(ptr, "Failed to find '__cxa_...');  
    return reinterpret_cast<void*(*)(std::size_t)>(ptr);  
}();
```

Сделаем подмешивание stacktrace к каждому throw!

– Создаём свою функцию `__cxa_allocate_exception`

– В ней:

• Получаем изначальный `__cxa_allocate_exception`

```
static const auto orig_allocate_exception = []() {  
    void* ptr = dlsym(RTLD_NEXT, "__cxa_allocate_exception");  
    BOOST_ASSERT_MSG(ptr, "Failed to find '__cxa_...');  
    return reinterpret_cast<void*(*)(std::size_t)>(ptr);  
}();
```

Сделаем подмешивание stacktrace к каждому throw!

– Создаём свою функцию `__cxa_allocate_exception`

– В ней:

• Получаем изначальный `__cxa_allocate_exception`

```
static const auto orig_allocate_exception = []() {  
    void* ptr = dlsym(RTLD_NEXT, "__cxa_allocate_exception");  
    BOOST_ASSERT_MSG(ptr, "Failed to find '__cxa_...');  
    return reinterpret_cast<void*(*)(std::size_t)>(ptr);  
}();
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```


Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

```
static constexpr std::size_t kAlign = alignof(std::max_align_t);  
thrown_size = (thrown_size + kAlign - 1) & (~(kAlign - 1));
```

```
constexpr std::size_t kStackTraceDumpSize = 4096;  
void* ptr = orig_allocate_exception(thrown_size + kStackTraceDumpSize);
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

```
char* dump_ptr = static_cast<char*>(ptr) + thrown_size;  
constexpr size_t kSkip = 1;  
trace::safe_dump_to(kSkip, dump_ptr, kStackTraceDumpSize);
```


Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

```
char* dump_ptr = static_cast<char*>(ptr) + thrown_size;  
constexpr size_t kSkip = 1;  
trace::safe_dump_to(kSkip, dump_ptr, kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

```
char* dump_ptr = static_cast<char*>(ptr) + thrown_size;  
constexpr size_t kSkip = 1;  
trace::safe_dump_to(kSkip, dump_ptr, kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

```
char* dump_ptr = static_cast<char*>(ptr) + thrown_size;  
constexpr size_t kSkip = 1;  
trace::safe_dump_to(kSkip, dump_ptr, kStackTraceDumpSize);
```

Сделаем подмешивание stacktrace к каждому throw!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

```
char* dump_ptr = static_cast<char*>(ptr) + thrown_size;  
constexpr size_t kSkip = 1;  
trace::safe_dump_to(kSkip, dump_ptr, kStackTraceDumpSize);
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде
 - Прикапываем указатель на `stacktrace`

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде
 - Прикапываем указатель на `stacktrace`
 - Возвращаем указатель на `storage` для тела исключения

Сделаем подмешивание `stacktrace` к каждому `throw`!

- Создаём свою функцию `__сха_allocate_exception`
- В ней:
 - Получаем изначальный `__сха_allocate_exception`
 - Подправляем размер для аллоцирования
 - Зовём изначальный метод с новым размером
 - Пишем туда `stacktrace` в компактном виде
 - Прикапываем указатель на `stacktrace`
 - Возвращаем указатель на `storage` для тела исключения
- Делаем метод для получения `stacktrace` из текущего исключения

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
namespace impl {  
const char* current_exception_stacktrace() noexcept {  
    auto exc_ptr = std::current_exception();  
    void* const exc_raw_ptr = get_current_exception_raw_ptr(&exc_ptr);  
    if (!exc_raw_ptr) {  
        return nullptr;  
    }  
    return reference_to_empty_padding(exc_raw_ptr);  
}  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```


Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```


Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Метод получения исключения

```
basic_stacktrace from_current_exception(Allocator alloc) noexcept {  
    const char* trace = impl::current_exception_stacktrace();  
    if (trace) {  
        try {  
            // Matches the constant from implementation  
            constexpr std::size_t kStackTraceDumpSize = 4096;  
            return from_dump(trace, kStackTraceDumpSize, alloc);  
        } catch (const std::exception&) {  
            // ignore  
        }  
    }  
    return basic_stacktrace{0, 0, alloc};  
}
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

```
try {  
    foo();  
} catch (const std::exception&) {  
    auto trace = boost::stacktrace::from_current_exception();  
    std::cout << "Trace: " << trace << '\n';  
}
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

```
try {  
    foo();  
} catch (const std::exception&) {  
    auto trace = boost::stacktrace::from_current_exception();  
    std::cout << "Trace: " << trace << '\n';  
}
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

```
try {  
    foo();  
} catch (const std::exception&) {  
    auto trace = boost::stacktrace::from_current_exception();  
    std::cout << "Trace: " << trace << '\n';  
}
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

```
try {  
    foo();  
} catch (const std::exception&) {  
    auto trace = boost::stacktrace::from_current_exception();  
    std::cout << "Trace: " << trace << '\n';  
}
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

Trace:

```
0# get_data_from_config(std::string_view) at /home/axo1m/basic.cpp:600
1# bar(std::string_view) at /home/axo1m/basic.cpp:6
2# foo() at /home/axo1m/basic.cpp:87
3# main at /home/axo1m/basic.cpp:17
```

Сделаем подмешивание `stacktrace` к каждому `throw`!

Trace:

```
0# get_data_from_config(std::string_view) at /home/axo1m/basic.cpp:600
1# bar(std::string_view) at /home/axo1m/basic.cpp:6
2# foo() at /home/axo1m/basic.cpp:87
3# main at /home/axo1m/basic.cpp:17
```


Сделаем подмешивание `stacktrace` к каждому `throw`!

Trace:

```
0# get_data_from_config(std::string_view) at /home/axo1m/basic.cpp:600
1# bar(std::string_view) at /home/axo1m/basic.cpp:6
2# foo() at /home/axo1m/basic.cpp:87
3# main at /home/axo1m/basic.cpp:17
```

Исключения медленные?

Исключения медленные?

Исключения медленные?

```
int nonthrowing_foo(int ) noexcept;
```

```
void without_exceptions(int i) {  
    // ...  
    if (!nonthrowing_foo(i)) {  
        // ...  
        __builtin_abort();  
    }  
}
```

```
void throwing_foo(int );
```

```
void with_exceptions(int i) {  
    // ...  
    throwing_foo(i);  
}
```

Исключения медленные?

```
int nonthrowing_foo(int ) noexcept;
```

```
void without_exceptions(int i) {  
    // ...  
    if (!nonthrowing_foo(i)) {  
        // ...  
        __builtin_abort();  
    }  
}
```

```
void throwing_foo(int );
```

```
void with_exceptions(int i) {  
    // ...  
    throwing_foo(i);  
}
```

Исключения медленные?

```
int nonthrowing_foo(int ) noexcept;
```

```
void without_exceptions(int i) {  
    // ...  
    if (!nonthrowing_foo(i)) {  
        // ...  
        __builtin_abort();  
    }  
}
```

```
void throwing_foo(int );
```

```
void with_exceptions(int i) {  
    // ...  
    throwing_foo(i);  
}
```

Исключения медленные?

```
int nonthrowing_foo(int ) noexcept;
```

```
void without_exceptions(int i) {  
    // ...  
    if (!nonthrowing_foo(i)) {  
        // ...  
        __builtin_abort();  
    }  
}
```

```
void throwing_foo(int );
```

```
void with_exceptions(int i) {  
    // ...  
    throwing_foo(i);  
}
```

Исключения медленные?

```
without_exceptions(int):  
    push    rax  
    call   nonthrowing_foo(int)  
    test   eax, eax  
    je     .LBB0_2  
    pop    rax  
    ret  
.LBB0_2:  
    call   abort@PLT
```

```
with_exceptions(int):  
    jmp    throwing_foo(int)
```


Исключения медленные?

```
without_exceptions(int):  
    push    rax  
    call   nonthrowing_foo(int)  
    test   eax, eax  
    je    .LBB0_2  
    pop    rax  
    ret  
  
.LBB0_2:  
    call   abort@PLT
```

```
with_exceptions(int):  
    jmp    throwing_foo(int)
```

Исключения медленные?

```
without_exceptions(int):  
    push    rax  
    call   nonthrowing_foo(int)  
    test   eax, eax  
    je     .LBB0_2  
    pop    rax  
    ret  
.LBB0_2:  
    call   abort@PLT
```

```
with_exceptions(int):  
    jmp    throwing_foo(int)
```

Исключения медленные!

Threads	1	2	4	8	16	32	64	128
0.0% failure	24ms	26ms	26ms	30ms	29ms	29ms	29ms	31ms
0.1% failure	29ms	29ms	29ms	29ms	30ms	30ms	31ms	105ms
1.0% failure	29ms	30ms	31ms	34ms	58ms	123ms	280ms	1030ms
10% failure	36ms	49ms	129ms	306ms	731ms	1320ms	2703ms	6425ms

<https://wg21.link/p2544>

Исключения медленные!

Исключения медленные!

- `dl_iterate_phdr` захватывает глобальный мьютекс

Исключения медленные!

- `dl_iterate_phdr` захватывает глобальный мьютекс
- `glibc 2.35` обзавёлся `_dl_find_object`

Исключения медленные!

- `dl_iterate_phdr` захватывает глобальный мьютекс
- `glibc 2.35` обзавёлся `_dl_find_object`
 - Clang-16 начал его использовать
 - GCC-13 начал его использовать

Исключения медленные!

- **dl_iterate_phdr** захватывает глобальный мьютекс
- glibc 2.35 обзавёлся `_dl_find_object`
 - Clang-16 начал его использовать
 - GCC-13 начал его использовать

Давайте подменим
`dl_iterate_phdr!`

Давайте подменим `dl_iterate_phdr`

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDLIteratePhdr(  
        [] (dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```


Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

```
std::atomic<PhdrCacheStorage*> phdr_cache_ptr{nullptr};
```

```
void Initialize() {  
    if (phdr_cache_ptr.load() != nullptr) { return; }  
  
    cache_.clear();  
    GetOriginalDlIteratePhdr()  
        [](dl_phdr_info* info, size_t /* size */, void* data) {  
            ((PhdrCacheStorage*)data)->push_back(*info);  
            return 0;  
        },  
        &cache_);  
    phdr_cache_ptr.exchange(&cache_);  
}
```

Давайте подменим dl_iterate_phdr

Но при этом *ломаются*:

- dlopen
- dlclose
- dlmopen

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```


Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
int dlclose(void *handle) {  
    using DLCloseSignature = int (*)(void*);  
    constexpr const char* kFunctionName = "dlclose";  
    static void* func = dlsym(RTLD_NEXT, "dlclose");  
    UASSERT(func);  
  
    AssertDynamicLoadingEnabled(kFunctionName);  
    return reinterpret_cast<DLCloseSignature>(func)(handle);  
}
```

Давайте подменим dl_iterate_phdr

```
void AssertDynamicLoadingEnabled(std::string_view dl_function_name) {  
    if (phdr_cache_ptr) {  
        const auto message = fmt::format(  
            "userver forbids '{} usage during components system lifetime due "  
            "to implementation details of making C++ exceptions scalable. You "  
            "may disable this optimization by either setting cmake option "  
            "USERVER_DISABLE_PHDR_CACHE or moving dynamic libraries "  
            "loading/unloading into components constructors/destructors.",  
            dl_function_name);  
        utils::impl::AbortWithStacktrace(message);  
    }  
}
```

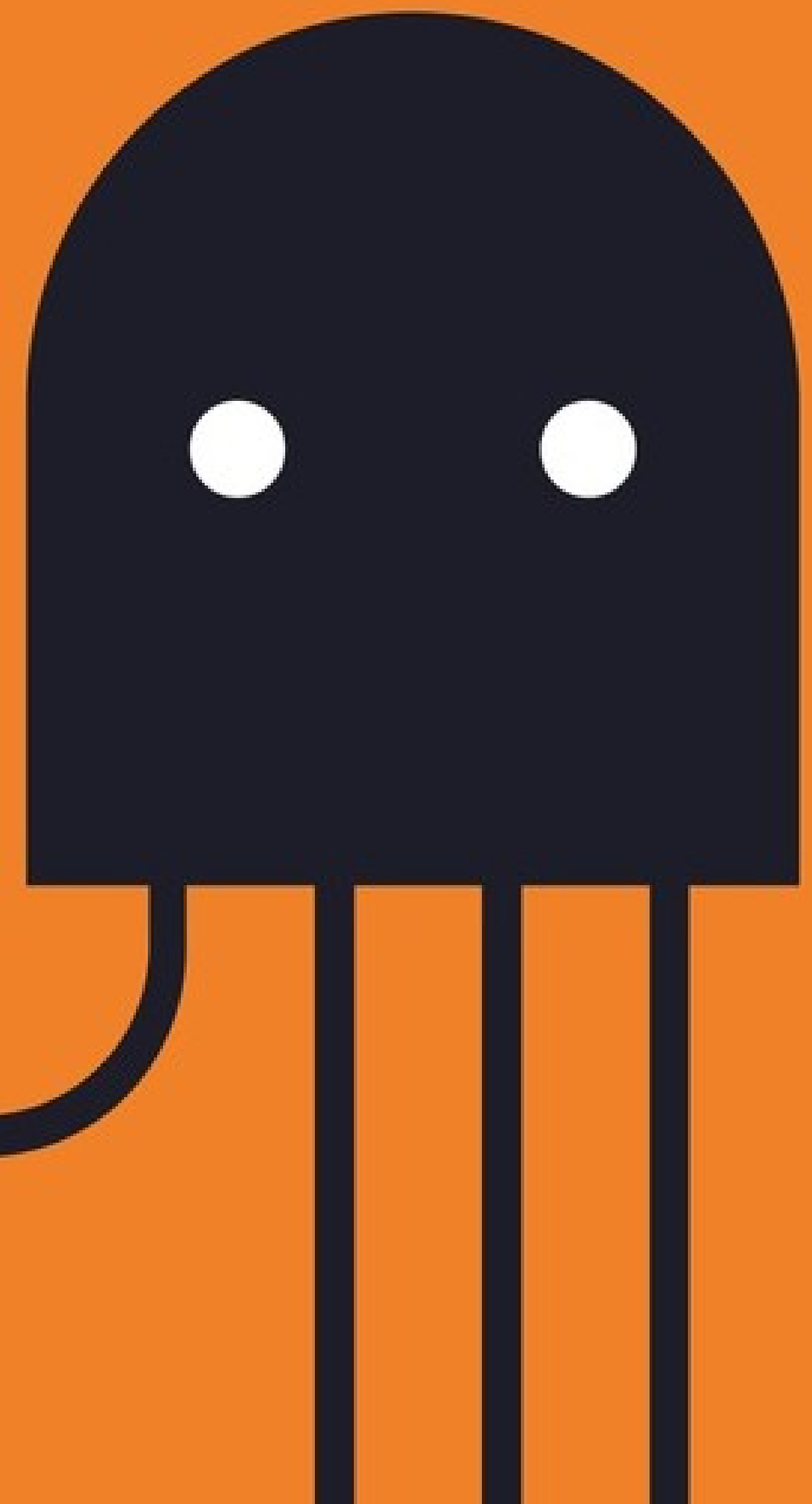
Давайте подменим dl_iterate_phdr

```
void AssertDynamicLoadingEnabled(std::string_view dl_function_name) {  
    if (phdr_cache_ptr) {  
        const auto message = fmt::format(  
            "userver forbids '{}' usage during components system lifetime due "  
            "to implementation details of making C++ exceptions scalable. You "  
            "may disable this optimization by either setting cmake option "  
            "USERVER_DISABLE_PHDR_CACHE or moving dynamic libraries "  
            "loading/unloading into components constructors/destructors.",  
            dl_function_name);  
        utils::impl::AbortWithStacktrace(message);  
    }  
}
```

Давайте подменим dl_iterate_phdr

```
void AssertDynamicLoadingEnabled(std::string_view dl_function_name) {  
    if (phdr_cache_ptr) {  
        const auto message = fmt::format(  
            "userver forbids '{}' usage during components system lifetime due "  
            "to implementation details of making C++ exceptions scalable. You "  
            "may disable this optimization by either setting cmake option "  
            "USERVER_DISABLE_PHDR_CACHE or moving dynamic libraries "  
            "loading/unloading into components constructors/destructors.",  
            dl_function_name);  
        utils::impl::AbortWithStacktrace(message);  
    }  
}
```

<https://userver.tech/>



Исключения медленные!

- `dl_iterate_phdr` захватывает глобальный мьютекс
- `glibc 2.35` обзавёлся `_dl_find_object`
 - Clang-16 начал его использовать
 - GCC-13 начал его использовать

~~Исключения медленные!~~

- ~~– `dl_iterate_phdr` захватывает глобальный мьютекс~~
- glibc 2.35 обзавёлся `_dl_find_object`
 - Clang-16 начал его использовать
 - GCC-13 начал его использовать

Загадка про 0x42 указатель

Загадка про 0x42 указатель

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```


Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_do_something() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_declval() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
// Structure that can be converted to reference to anything
struct ubiq_lref_constructor {
    template <class Type>
    constexpr operator Type&() const & noexcept {
        return unsafe_declval<Type&>();
    }
};
```

Загадка про 0x42 указатель

```
// Structure that can be converted to reference to anything
struct ubiq_lref_constructor {
    template <class Type>
    constexpr operator Type&() const & noexcept {
        return unsafe_declval<Type&>();
    }
};
```

Загадка про 0x42 указатель

```
// Structure that can be converted to reference to anything
struct ubiq_lref_constructor {
    template <class Type>
    constexpr operator Type&() const & noexcept {
        return unsafe_declval<Type&>();
    }
};
```


Загадка про 0x42 указатель

```
// Structure that can be converted to reference to anything
struct ubiq_lref_constructor {
    template <class Type>
    constexpr operator Type&() const & noexcept {
        return unsafe_declval<Type&>();
    }
};
```

Загадка про 0x42 указатель

```
// Structure that can be converted to reference to anything
struct ubiq_lref_constructor {
    template <class Type>
    constexpr operator Type&() const & noexcept {
        return unsafe_declval<Type&>(); // GCCs std::declval may not be
        // used in potentially evaluated contexts,
        // so we reinvent it.
    }
};
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_declval() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_declval() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
template <class T>
constexpr T unsafe_declval() noexcept {
    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
// This function serves as a link-time assert. If linker requires it, then
// `unsafe_declval()` is used at runtime.
void report_if_you_see_link_error_with_this_function() noexcept;

template <class T>
constexpr T unsafe_declval() noexcept {
    report_if_you_see_link_error_with_this_function();

    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```

Загадка про 0x42 указатель

```
// This function serves as a link-time assert. If linker requires it, then  
// `unsafe_declval()` is used at runtime.
```

```
void report_if_you_see_link_error_with_this_function() noexcept;
```

```
template <class T>
```

```
constexpr T unsafe_declval() noexcept {
```

```
    report_if_you_see_link_error_with_this_function();
```

```
    typename std::remove_reference<T>::type* ptr = nullptr;
```

```
    ptr += 42;
```

```
    return static_cast<T>(*ptr);
```

```
}
```

Загадка про 0x42 указатель

```
// This function serves as a link-time assert. If linker requires it, then
// `unsafe_declval()` is used at runtime.
void report_if_you_see_link_error_with_this_function() noexcept;

template <class T>
constexpr T unsafe_declval() noexcept {
    report_if_you_see_link_error_with_this_function();

    typename std::remove_reference<T>::type* ptr = nullptr;
    ptr += 42;
    return static_cast<T>(*ptr);
}
```


А что будет если...

А что будет если...

А что будет если...

```
#include <iostream>
```

```
template <auto member_ptr>
```

```
void print() {
```

```
    std::cout << __PRETTY_FUNCTION__ << std::endl;
```

```
}
```

А что будет если...

```
#include <iostream>
```

```
template <auto member_ptr>
```

```
void print() {
```

```
    std::cout << __PRETTY_FUNCTION__ << std::endl;
```

```
}
```

А что будет если...

```
#include <iostream>
```

```
template <auto member_ptr>
```

```
void print() {
```

```
    std::cout << __PRETTY_FUNCTION__ << std::endl;
```

```
}
```

А что будет если...

```
#include <iostream>
```

```
template <auto member_ptr>
```

```
void print() {
```

```
    std::cout << __PRETTY_FUNCTION__ << std::endl;
```

```
}
```

А что будет если...

```
#include <iostream>
```

```
template <auto member_ptr>
```

```
void print() {
```

```
    std::cout << __PRETTY_FUNCTION__ << std::endl;
```

```
}
```

А что будет если...

```
#include <iostream>

template <auto member_ptr>
void print() {
    std::cout << __PRETTY_FUNCTION__ << std::endl;
}

struct S {
    int the_member_name;
} s;

int main() { print<&s.the_member_name>(); }
```


А что будет если...

```
#include <iostream>

template <auto member_ptr>
void print() {
    std::cout << __PRETTY_FUNCTION__ << std::endl;
}

struct S {
    int the_member_name;
} s;

int main() { print<&s.the_member_name>(); }
```

А что будет если...

```
#include <iostream>

template <auto member_ptr>
void print() {
    std::cout << __PRETTY_FUNCTION__ << std::endl;
}

struct S {
    int the_member_name;
} s;

int main() { print<&s.the_member_name>(); }
```

А что будет если...

```
void print() [with auto member_ptr = (& s.S::the_member_name)]
```

Получение имён полей структуры

```
void print() [with auto member_ptr = (& s.S::the_member_name)]
```

Получение имён полей структуры

```
void print() [with auto member_ptr = (& s.S::the_member_name)]
```

Финальный шаг

Финальный шаг

```
struct S2 {  
    int the_member_name;  
    short other_name;  
} s2;
```

Финальный шаг

```
struct s2 {  
    int the_member_name;  
    short other_name;  
} s2;  
  
int main() {  
    const auto& [a, b] = s2;  
    print<&a>();  
    print<&b>();  
}
```


Финальный шаг

```
struct s2 {  
    int the_member_name;  
    short other_name;  
} s2;  
  
int main() {  
    const auto& [a, b] = s2;  
    print<&a>();  
    print<&b>();  
}
```

Финальный шаг

```
struct s2 {  
    int the_member_name;  
    short other_name;  
} s2;  
  
int main() {  
    const auto& [a, b] = s2;  
    print<&a>();  
    print<&b>();  
}
```

Финальный шаг

```
struct s2 {  
    int the_member_name;  
    short other_name;  
} s2;  
  
int main() {  
    const auto& [a, b] = s2;  
    print<&a>();  
    print<&b>();  
}
```

Получение имён полей структуры, финал

```
void print() [member_ptr = &s2.the_member_name]
```

```
void print() [member_ptr = &s2.other_name]
```

Получение имён полей структуры, финал

```
void print() [member_ptr = &s2.the_member_name]
```

```
void print() [member_ptr = &s2.other_name]
```

Получение имён полей структуры, финал

```
void print() [member_ptr = &s2.the_member_name]
```

```
void print() [member_ptr = &s2.other_name]
```

Финальный шаг

```
struct sample {  
    int f_int;  
    long f_long;  
};
```

```
std::cout << boost::pfr::get_name<0, sample>() // f_int  
          << boost::pfr::get_name<1, sample>(); // f_long
```

Спасибо

Полухин Антон

Эксперт-разработчик C++



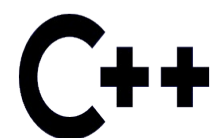
antoshkka@gmail.com



antoshkka@yandex-team.ru

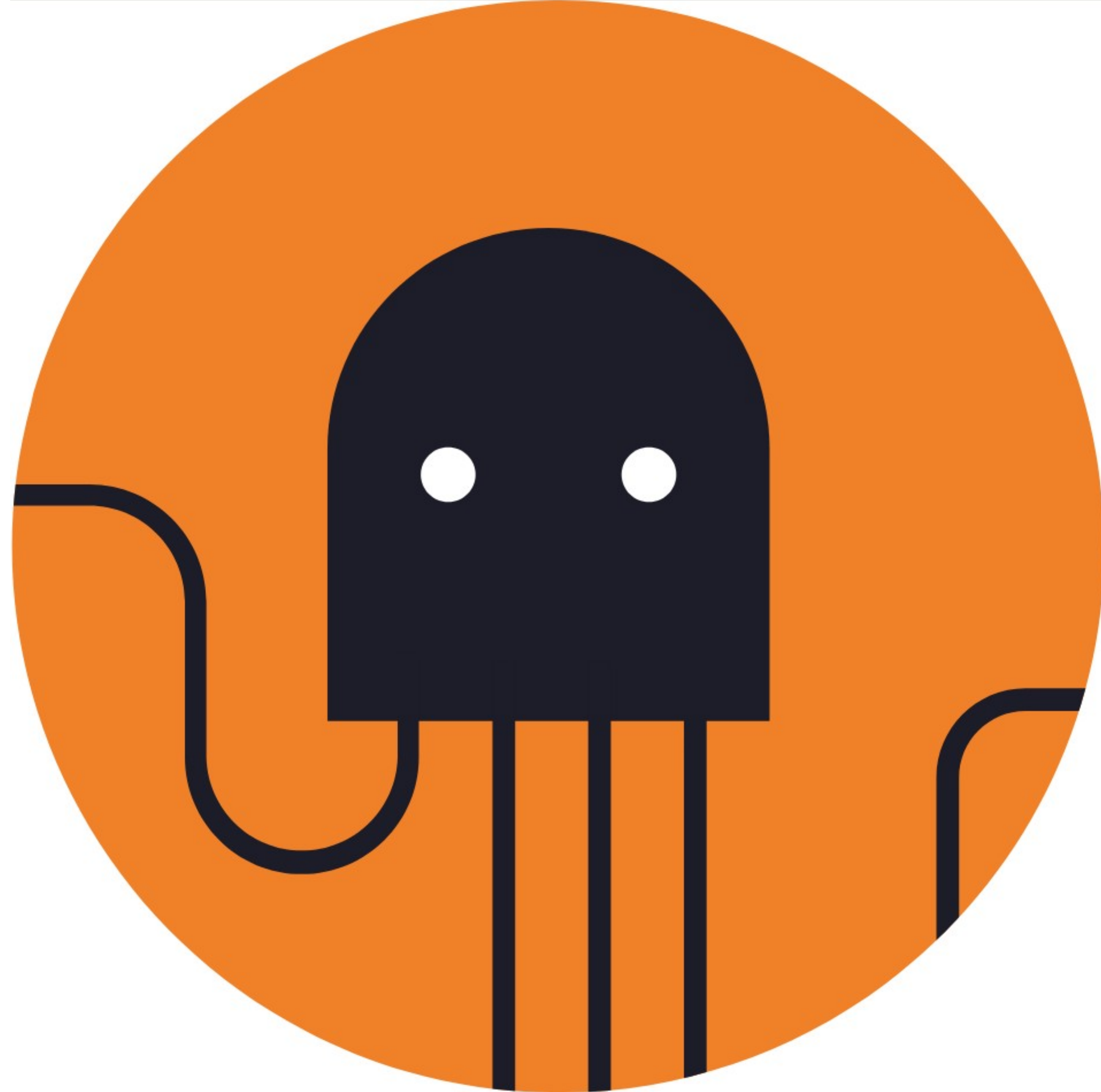


<https://github.com/apolukhin>



<https://stdcpp.ru/>

РГ21 C++ РОССИЯ



<https://github.com/userver-framework>