

Windows, MacOS and the Web: Lessons from Cross-platform Development at think-cell

Sebastian Theophil

C++ now

2021
MAY 2-7

Aspen, Colorado, USA

The Problem

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded

The Problem

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)

The Problem

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)
 - Render into application-supplied objects:
 - NSView, CALayer, HWND, DirectX texture
 - Renderer needs to support DirectX and OpenGL/Metal
 - Share the main message loop
 - Support platform-specific features like host application

The Problem

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)
 - Render into application-supplied objects:
 - NSView, CALayer, HWND, DirectX texture
 - Renderer needs to support DirectX and OpenGL/Metal
 - Share the main message loop
 - Support platform-specific features like host application
- Need cross-platform toolkit that hides platform specifics and **behaves identically** on different platforms

1. **Levels of Abstraction: Hiding Platform-Specifics**
2. Kernel Object Lifetimes: Interprocess Shared Memory
3. Common Tooling I: Text Internationalization
4. Common Tooling II: Error Reporting
5. Moving to WebAssembly

Platform-independent C++

?

Operating System

Platform-independent C++

?

Operating System

That depends ... on the semantics of your code!

Easy cases with clear semantics:

- Rendering engine with DirectX/OpenGL backend
- HTTP requests using system APIs (WinINet, NSURLConnection)
- Start child process and setup I/O pipes

Easy cases with clear semantics:

- Rendering engine with DirectX/OpenGL backend
- HTTP requests using system APIs (WinINet, NSURLConnection)
- Start child process and setup I/O pipes

Clearly defined data in, data out

Levels of Abstraction

Tricky case: Direct calls to OS, e.g. renaming a file

```
BOOL MoveFileExW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, DWORD dwFlags)
```

`MOVEFILE_COPY_ALLOWED` Allow copy when moving to different volume

`MOVEFILE_REPLACE_EXISTING` Overwrite target

Levels of Abstraction

Tricky case: Direct calls to OS, e.g. renaming a file

```
BOOL MoveFileExW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, DWORD dwFlags)
```

`MOVEFILE_COPY_ALLOWED` Allow copy when moving to different volume

`MOVEFILE_REPLACE_EXISTING` Overwrite target

```
int renamex_np(char const* from, char const* to, unsigned int flags) // darwin  
int renameat2(int olldirfd, const char *oldpath,  
             int newdirfd, const char *newpath, unsigned int flags) // Linux
```

`RENAME_SWAP/RENAME_EXCHANGE` Swap source and target atomically on supported file systems

`RENAME_EXCL/RENAME_NOREPLACE` Fail if destination already exists on supported file systems

Levels of Abstraction

Tricky case: Direct calls to OS, e.g. renaming a file

```
int copyfile(char const* from, char const* to, copyfile_state_t state,  
            copyfile_flags_t flags)
```

`COPYFILE_ACL` Copy the source file's access control lists.

`COPYFILE_DATA` Copy the source file's data.

`COPYFILE_EXCL` Fail if the to file already exists

`COPYFILE_MOVE` Unlink source

Levels of Abstraction

Tricky case: Direct calls to OS, e.g. renaming a file

```
int copyfile(char const* from, char const* to, copyfile_state_t state,  
            copyfile_flags_t flags)
```

`COPYFILE_ACL` Copy the source file's access control lists.

`COPYFILE_DATA` Copy the source file's data.

`COPYFILE_EXCL` Fail if the to file already exists

`COPYFILE_MOVE` Unlink source

How to write an abstraction for that?

Levels of Abstraction

Tricky case: Direct calls to OS, e.g. renaming a file

```
int copyfile(char const* from, char const* to, copyfile_state_t state,  
            copyfile_flags_t flags)
```

`COPYFILE_ACL` Copy the source file's access control lists.

`COPYFILE_DATA` Copy the source file's data.

`COPYFILE_EXCL` Fail if the to file already exists

`COPYFILE_MOVE` Unlink source

How to write an abstraction for that?

```
void std::filesystem::rename(path const&, path const& new_p)
```

How can we do better?



Fabian Giesen
@rygorous

...

Pet peeve: code chock full of `#ifdefs` is _not_ "portable code". It is, at best, code that's been ported a lot. There's a big difference.

[Tweet übersetzen](#)

8:25 nachm. · 3. Feb. 2020 · Twitter Web App

34 Retweets 5 Zitierte Tweets 288 „Gefällt mir“-Angaben

- Hard to get identical **and useful** low level semantics on different platforms

- Hard to get identical **and useful** low level semantics on different platforms
- We don't need a cross-platform file rename function!
- **This is the wrong level of abstraction!**
- Need functions to

- Hard to get identical **and useful** low level semantics on different platforms
- We don't need a cross-platform file rename function!
- **This is the wrong level of abstraction!**
- Need functions to
 - download a file to a cache in thread-safe way
 - create a user application settings file
 - create a temporary file that is automatically deleted but which can be opened by another application
 - create a document in a user-specified folder,
open system-specific “Save File” dialog,
and create sandbox exception
- Functions with **strong** and **identical** semantics

Levels of Abstraction

- Create a user application settings file
 - Windows: `%APPDATA%\think-cell [+ integrity level]\`
 - macOS: `~/Library/Application Support/think-cell/` or
`~/Library/Group Containers/[Application Group Identifier]/`
 - Exclusive access while writing
 - Inherit ACL from parent folder

Levels of Abstraction

- Create a temporary file that is automatically deleted but which can be opened by another application

```
CreateFile(  
    "%TEMP%\...\",  
    FILE_GENERIC_READ|FILE_GENERIC_WRITE,  
    FILE_SHARE_READ,  
    // make SECURITY_ATTRIBUTES FILE_ALL_ACCESS&~FILE_EXECUTE  
    // accessible by current user only,  
    CREATE_NEW, // the file should not already exist  
    FILE_ATTRIBUTE_TEMPORARY | FILE_ATTRIBUTE_HIDDEN  
    | FILE_FLAG_DELETE_ON_CLOSE,  
    nullptr  
);  
// synchronize file access
```

Levels of Abstraction

- Create a temporary file that is automatically deleted but which can be opened by another application

```
open(
    "~/Library/Group Containers/[Application Group Identifier]/...",
    O_RDWR | O_CREAT | O_EXCL
    | O_NOFOLLOW | O_CLOEXEC
    | O_SHLOCK,
    S_IRUSR|S_IWUSR
);
// synchronize file access
// handle EINTR
// manual reference counting in shared memory
```

- Create a temporary file that is automatically deleted but which can be opened by another application

```
open(
    "~/Library/Group Containers/[Application Group Identifier]/...",
    O_RDWR | O_CREAT | O_EXCL
    | O_NOFOLLOW | O_CLOEXEC
    | O_SHLOCK,
    S_IRUSR|S_IWUSR
);
// synchronize file access
// handle EINTR
// manual reference counting in shared memory
```

Alternative:

Store temporary data in SQLite database and delete database on shutdown

- Cross-platform interfaces need to have well-defined, strong semantics
- Weak semantics lead to subtle errors
 - Warning sign: Having to look at the implementation
- Strong semantics increase DOF for implementor

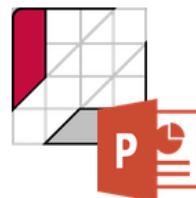
- Cross-platform interfaces need to have well-defined, strong semantics
- Weak semantics lead to subtle errors
 - Warning sign: Having to look at the implementation
- Strong semantics increase DOF for implementor
- Too high-level:
 - Missed chance to unify code. Rare. We are lazy.
- Too low-level:
 - You'll force identical interfaces on very different things
 - Semantics don't match operating system (`QFile::setPermissions`)
 - or you lose a lot of expressiveness (`rename`)

1. Levels of Abstraction: Hiding Platform-Specifics
2. **Kernel Object Lifetimes: Interprocess Shared Memory**
3. Common Tooling I: Text Internationalization
4. Common Tooling II: Error Reporting
5. Moving to WebAssembly

- We use shared memory to implement inter-process communication
- Boost.Interprocess (like Qt) offers a common API for shared memory on Windows and Posix



gettyimages®
OAuth 2.0



- We use shared memory to implement inter-process communication
- Boost.Interprocess (like Qt) offers a common API for shared memory on Windows and Posix
- named memory objects mappable into different processes
 - `boost::interprocess::managed_shared_memory`
- pointers stored in shared memory
 - `boost::interprocess::offset_ptr`
 - store offset to their own this pointer
 - shared memory can be mapped at different addresses
- named synchronization objects
 - `boost::interprocess::named_mutex`

Server process:

```
#include <boost/interprocess/managed_shared_memory.hpp>
...
using namespace boost::interprocess;
using T = std::pair<double, int>

// Name specific to build version, host, security identifier
char const szName[] = { ... };
managed_shared_memory seg(create_only, szName, /* size */ 65536);
T* p = seg.construct<T>("A") // name of the object
(10.0, 0); // ctor first argument
```

Child process:

```
// Open managed shared memory
managed_shared_memory seg(open_only, szName);

// Find object
T* p = seg.find<T>("A").first;
assert(p->first == 10.0);
```

Server process:

```
#include <boost/interprocess/managed_shared_memory.hpp>
...
using namespace boost::interprocess;
using T = std::pair<double, int>

// Name specific to build version, host, security identifier
char const szName[] = { ... };
managed_shared_memory seg(create_only, szName, /* size */ 65536);
T* p = seg.construct<T>("A") // name of the object
    (10.0, 0); // ctor first argument
```

Child process:

```
// Open managed shared memory
managed_shared_memory seg(open_only, szName);

// Find object
T* p = seg.find<T>("A").first;
assert(p->first == 10.0);
```

- But `managed_shared_memory` does not use native Windows shared memory

```
HANDLE hMapFile = CreateFileMapping(
    /* use page file */ INVALID_HANDLE_VALUE,
    /* security attributes */ NULL,
    PAGE_READWRITE,
    0,
    /* size */ 256,
    /* object name */ szName);
```

- On Windows, if last process accessing the named memory segment dies, memory is freed
- **This is a feature you want**

- But `managed_shared_memory` does not use native Windows shared memory

```
HANDLE hMapFile = CreateFileMapping(
    /* use page file */ INVALID_HANDLE_VALUE,
    /* security attributes */ NULL,
    PAGE_READWRITE,
    0,
    /* size */ 256,
    /* object name */ szName);
```

- On Windows, if last process accessing the named memory segment dies, memory is freed
- **This is a feature you want**
... and not supported on Posix

- This seems to cause problems occasionally:

The screenshot shows a Stack Overflow question page. The left sidebar includes links for Home, PUBLIC, Stack Overflow (which is selected), Tags, Users, FIND A JOB (Jobs and Companies), TEAMS, and a Free 30 Day Trial offer. The main content area displays a question titled "Remove posix shared memory when not in use?" asked 7 years, 11 months ago, with 15 answers, 3 comments, and 8k views. The top answer discusses the difficulty of automatically removing shared memory segments in Linux.

Remove posix shared memory when not in use?

Asked 7 years, 11 months ago Active 2 months ago Viewed 8k times

15 Is there any way, linux specific or not, to have posix shared memory segments (obtained with `shm_open()`) removed when no process is using them. i.e. have them reference counted and have the system remove them when the reference becomes 0

A few notes:

- Establishing an atexit handler to remove them doesn't work if the program crashes.
- Currently, the linux specific way, I embed the pid in the segment name, and try to find unused segments by walking /dev/shm in an external program. Which has the drawback of having to periodically clean them up externally in a rather hackish way.
- As the program can run multiple copies, using a well defined name for the segment that the program reuses when it starts up is not feasible.

- This seems to cause problems occasionally:

The screenshot shows a Stack Overflow question page. The question title is "Mutex in shared memory when one user crashes?". It was asked 10 years, 11 months ago and has been active for 8 years, 8 months ago. It has been viewed 10k times. The question text asks how to detect a mutex is already locked but not owned by any process. It has 15 upvotes and 6 answers. Tags listed are c++, c, linux, mutex, shared-memory. The question was edited on Dec 2 '11 at 17:27 by Zan Lynx. A comment from Vivek is visible, asking if the question is still relevant. The sidebar on the left includes links for Home, PUBLIC, Stack Overflow (selected), Tags, Users, FIND A JOB, Jobs, Companies, TEAMS, and a Free 30 Day Trial.

stackoverflow Products Search...

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS

What's this?

Free 30 Day Trial

Mutex in shared memory when one user crashes?

Asked 10 years, 11 months ago Active 8 years, 8 months ago Viewed 10k times

15

Suppose that a process is creating a mutex in shared memory and locking it and dumps core while the mutex is locked.

Now in another process how do I detect that mutex is already locked but not owned by any process?

6 c++ c linux mutex shared-memory

share edit follow close flag

edited Dec 2 '11 at 17:27

Zan Lynx

47.1k 7 73 122

Vivek

423 1 3 10

add a comment

start a bounty

- This seems to cause problems occasionally:

boostorg / interprocess

Code Issues 38 Pull requests 15 Actions Projects Wiki Security Insights

robustness of the interprocess mutex #65

reed-lau opened this issue on 26 Oct 2018 · 0 comments

reed-lau commented on 26 Oct 2018

when using interprocess shared memroy, there are about two locks. one is the internal mutex_family used by boost for managing the sharedmemory in MemoryAlgorithm, one is the user's one for synchronization. both mutex will cause deadlock, when process crash. How do you think about it.

the linux provide pthread_mutexattr_setrobust, but the interprocess module do not provide the interface, eg. interprocess::scoped_lock's lock's return value is void instead of int, which could be compare with EOWNERDEAD.

- This seems to cause problems occasionally:

The screenshot shows a Stack Overflow question page. The left sidebar includes links for Home, PUBLIC, Stack Overflow (selected), Tags, Users, FIND A JOB, Jobs, Companies, TEAMS, and a Free 30 Day Trial. The main content area displays a question titled "how to free managed_shared_memory after a program crash: what are effective techniques to use during debugging?". The question was asked 4 years, 4 months ago and has been active for 4 years, 3 months ago, with 460 views. The first answer, by user '1', has an upvote icon and the text: "Admittedly I am a novice and self-taught programmer, and am finally venturing into the depths and power of C and C++. Some things that come with this self-learning process are not textbook or overtly googleable knowledge, such as tricks to use in difficult circumstances, and debugging strategies." Below it is another answer with a downvote icon and the text: "I am using boost::interprocess to set up shared memory using the managed_shared_memory and named objects. Unsurprisingly, my program crashes here and there for a number of reasons during my development. So far, I am rather unfamiliar with debugging tools, especially in linux. As a consequence, many times my shared memory does not get removed properly, as the crashing can result in destructors never being called, etc." At the bottom, a note states: "So, after such a program crash, when I attempt to run my application again, when my code tries to allocate a new segment of shared memory I see messages like:".

- This seems to cause problems occasionally:

[Boost-users] [interprocess] named mutex clean up 90 views



Chard

to boost...@lists.boost.org

Does calling named_mutex::remove() have consistent cross-platform behaviour?

The reason I ask is that I am trying to use a named mutex for an "I'm the only process" check.

That is, the process takes a shared lock on the named mutex (at start up), then, at points within the program, it attempts to get an exclusive lock in order to perform the check/actions. When finished, the mutex is returned to a shared lock.

— · · · · —

- On Windows, shared memory and mutexes are reference counted,
i.e., when the last user dies or crashes, resource is freed
- On Posix, shared memory is either
 1. File-backed:
backing file still exists if processes crash, even after reboot
 2. Posix shared memory (`shm_open` / `shm_unlink`):
backing memory persists until reboot
- Posix model assumes server-client model
& Server owns shared memory object
- If not, there are two solutions to this problem.
Boost.Interprocess and Qt implement **neither**

1. Robust mutexes

```
pthread_mutexattr_t attr;
pthread_mutexattr_init(&attr);
// Allow mutex to be placed in shared memory
pthread_mutexattr_setpshared(&attr, PTHREAD_PROCESS_SHARED);
// Mark mutex as robust
pthread_mutexattr_setrobust(&attr, PTHREAD_MUTEX_ROBUST);

pthread_mutex_t mtx;
pthread_mutex_init(&mtx, &attr);

[...]

if(EOWNERDEAD == pthread_mutex_lock(&mtx)) {
    // Owner of lock has died, reinitialize shared memory
}
```

See Linux man pages and ❤ this Boost.Interprocess pull request <https://github.com/boostorg/interprocess/pull/67>

2. File Locking

- macOS does not support robust pthread locks unfortunately
- the only other resource that has process-lifetime are file locks
- **Attention!** File locks are weird on Posix as well:
"Everything you never wanted to know about file locking" <https://apenwarr.ca/log/20101213>
- Please  our Boost.Interprocess pull request <https://github.com/boostorg/interprocess/pull/132>

- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, 0_CREAT | 0_EXLOCK | 0_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, 0_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request <https://github.com/boostorg/interprocess/pull/132>

- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, 0_CREAT | 0_EXLOCK | 0_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, 0_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request <https://github.com/boostorg/interprocess/pull/132>

- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, 0_CREAT | 0_EXLOCK | 0_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, 0_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request <https://github.com/boostorg/interprocess/pull/132>

- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, 0_CREAT | 0_EXLOCK | 0_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, 0_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request <https://github.com/boostorg/interprocess/pull/132>

- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, 0_CREAT | 0_EXLOCK | 0_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, 0_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request <https://github.com/boostorg/interprocess/pull/132>

- Aiming for strong and identical semantics
 - ... strong semantics means strong guarantees!
 - ... don't sacrifice operating system guarantees for identical API!
- Implementing strong & identical semantics may be hard
 - ... and sometimes you can do better than existing toolkits!
 - ... because you know your constraints

1. Levels of Abstraction: Hiding Platform-Specifics
2. Kernel Object Lifetimes: Interprocess Shared Memory
3. **Common Tooling I: Text Internationalization**
4. Common Tooling II: Error Reporting
5. Moving to WebAssembly

- Internationalization is more than translation, but focus on that today
- Important translation features:

1. Annotate translatable text in code

```
TRANSLATE("Do not ask for user name/password")
```

2. Translation context

```
TRANSLATECTX("Proxy Authentication",
"Proxy: http://en.wikipedia.org/wiki/Proxy_server.")
```

3. There are arbitrary number of plural forms

0 weeks - 0 недель

1 week - 1 неделя

4 weeks - 4 недели

5 weeks - 5 недель

Text Internationalization

- What is the general flow for i18n:
 1. Annotation in source code
 2. Extraction of translatable text
 3. Send to translators
 4. Get `xliff` (XML Localization Interchange File Format) file
 5. Import into project as resource
 6. At program runtime, lookup text/language pair
- Supporting native mechanisms would suck
 - You want the same markup in code
 - You want a single text extraction run, preferably platform independent
 - You want a uniform access mechanism for translatable strings, no lifetime issues (`char const*`!)
- `Boost.Locale` would fit the bill

Text Internationalization

- Boost.Locale was added 2018 in boost 1.67:

```
std::cout << translate("Hello World") << std::endl;
```

"Привет мир"

```
std::cout << translate("File", "open") << std::endl;
```

"öffnen"

```
std::cout << format(translate("You have {1} file in the directory",
                             "You have {1} files in the directory",
                             3)) % 3 << std::endl;
```

"У вас есть 3 файла в каталоге"

Text Internationalization

- Boost.Locale was added 2018 in boost 1.67:

```
std::cout << translate("Hello World") << std::endl;
```

"Привет мир"

```
std::cout << translate("File", "open") << std::endl;
```

"öffnen"

```
std::cout << format(translate("You have {1} file in the directory",
                             "You have {1} files in the directory",
                             3)) % 3 << std::endl;
```

"У вас есть 3 файла в каталоге"

- We don't have to do runtime text lookups like this is 1995
- We have `constexpr` functions!

Text Internationalization

```
#include <array>

enum ESupportedLanguage {
    elangEN, elangRU, elangCOUNT
};

using STranslatableString = std::array<char const*, elangCOUNT>;

char const* translate(STranslatableString ts) noexcept;

#define TRANSL0OKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSL0OKUP(String, Context))

int main() noexcept {
    std::cout
        << TRANSLATECTX("Cancel", "Cancel: as in Windows dialogs.")
        << std::endl;
}
```

```
#include <array>

enum ESupportedLanguage {
    elangEN, elangRU, elangCOUNT
};

using STranslatableString = std::array<char const*, elangCOUNT>;

char const* translate(STranslatableString ts) noexcept;

#define TRANSL0OKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSL0OKUP(String, Context))

int main() noexcept {
    std::cout
        << TRANSLATECTX("Cancel", "Cancel: as in Windows dialogs.")
        << std::endl;
}
```

Text Internationalization

```
#include <array>

enum ESupportedLanguage {
    elangEN, elangRU, elangCOUNT
};

using STranslatableString = std::array<char const*, elangCOUNT>;

char const* translate(STranslatableString ts) noexcept;

#define TRANSL0OKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSL0OKUP(String, Context))

int main() noexcept {
    std::cout
        << TRANSLATECTX("Cancel", "Cancel: as in Windows dialogs.")
        << std::endl;
}
```

```
using STranslatableString = std::array<char const*, elangCOUNT>;  
  
template<std::uint64_t, std::uint64_t, std::uint64_t, std::uint64_t>  
struct STranslatableStringMap {  
    static STranslatableString const m_apsz;  
};  
  
#define TRANSLOOKUP(String, Context) \  
    (STranslatableStringMap< \  
        StaticMurmurHash::Hash(u8 ## String).first, \  
        StaticMurmurHash::Hash(u8 ## String).second, \  
        StaticMurmurHash::Hash(u8 ## Context).first, \  
        StaticMurmurHash::Hash(u8 ## Context).second \  
    >::m_apsz)
```

Text Internationalization

```
using STranslatableString = std::array<char const*, elangCOUNT>;  
  
template<std::uint64_t, std::uint64_t, std::uint64_t, std::uint64_t>  
struct STranslatableStringMap {  
    static STranslatableString const m_apsz;  
};  
  
template<> constexpr std::array<char const*, elangCOUNT>  
STranslatableStringMap<0x17aa19f18a894459, 0xab6ff21156e8a341,  
    0x691fc12842a48f3c, 0x4720a1a3af148ae1  
>::m_apsz{  
    "Cancel", "Отменить"  
};  
  
#define TRANSLOOKUP(String, Context) \  
    (STranslatableStringMap< \  
        StaticMurmurHash::Hash(u8 ## String).first, \  
        StaticMurmurHash::Hash(u8 ## String).second, \  
        StaticMurmurHash::Hash(u8 ## Context).first, \  
        StaticMurmurHash::Hash(u8 ## Context).second \  
    >::m_apsz)
```

- Reminder constexpr functions:
 - implicitly inline.
 - must accept and return only literal types.
 - i.e. scalars, references,
 - aggregate types `T t = { ... };`, e.g., `std::array`
 - type with `constexpr` ctor and dtor, all data members and base classes are literal types, e.g., `std::pair`
 - arrays of such types
 - can't be virtual, cannot contain `goto` or `try/catch`
 - may contain `if`, `switch`, all loop statements
 - local variable declarations if variable is initialized and literal type
 - ctor cannot be `constexpr` if class has virtual base classes

Text Internationalization

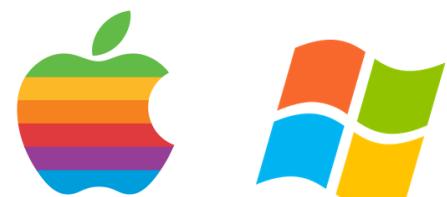
```
namespace StaticMurmurHash {
    using uint128_t = std::pair<std::uint64_t, std::uint64_t>;
    template<int N>
    constexpr uint128_t Hash(
        char const (&at)[N],
        std::uint32_t seed=0) noexcept
    {
        return MurmurHash3_x64_128(&at[0], N, seed);
    }
}
```

See <https://github.com/aappleby/smhasher> for MurmurHash sources

```
namespace StaticMurmurHash {
    using uint128_t = std::pair<std::uint64_t, std::uint64_t>;
    template<int N>
    constexpr uint128_t Hash(
        char const (&at)[N],
        std::uint32_t seed=0) noexcept
    {
        return MurmurHash3_x64_128(&at[0], N, seed);
    }
}
```

See <https://github.com/aappleby/smhasher> for MurmурHash sources

- Aiming for strong and identical semantics
... that includes external tools in your build process!



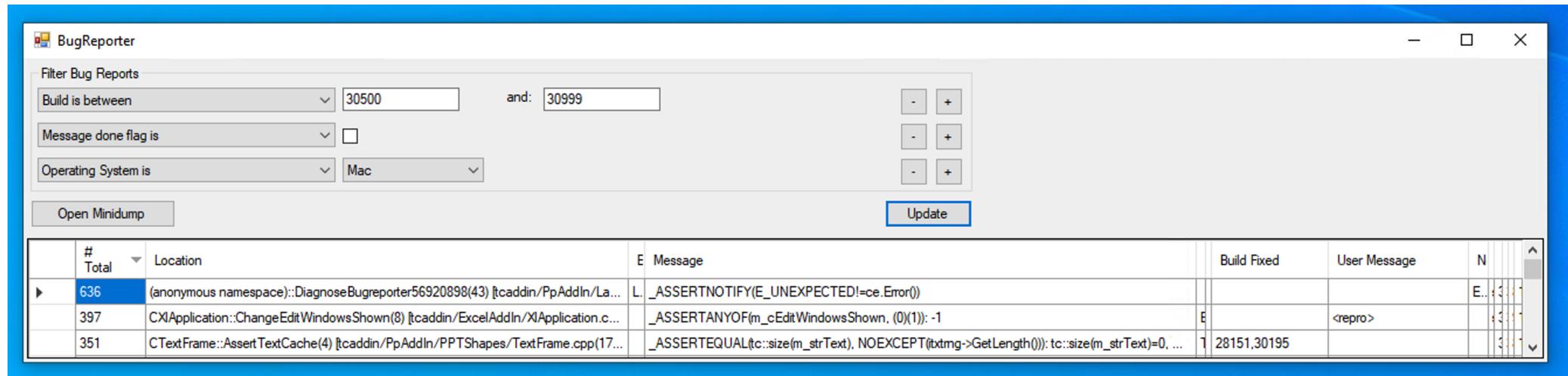
2 : 1

1. Levels of Abstraction: Hiding Platform-Specifics
2. Kernel Object Lifetimes: Interprocess Shared Memory
3. Common Tooling I: Text Internationalization
- 4. Common Tooling II: Error Reporting**
5. Moving to WebAssembly

Error Reporting

- think-cell has a powerful error reporting architecture
- We use a lot of ASSERTs to check invariants
- Check all system API error codes, distinguish expected and unexpected errors
- Once we encounter unexpected behavior
 1. Decide to show user message
 2. Send an error report home
 - Our backend analyzes error report
 - Checks if we have already fixed bug or if we would like more info from user
 - Reports back
 - May silently download & install update

Error Reporting



- Developers check and analyze most frequent bug reports regularly
 - Annotate in which version bug is fixed
- Backend pre-analyzes bug reports:
 - Walk stack back to relevant frame (skip smart pointers, error reporting code)
 - Group errors based on method offset (not source line, nor error message)
- **Finds a lot of bugs that depend on user's setup**

The core of this functionality on Windows:

```
BOOL MiniDumpWriteDump(  
    HANDLE hProcess,          // handle to process  
    DWORD ProcessId,         // process ID  
    HANDLE hFile,             // handle to file  
    MINIDUMP_TYPE DumpType,  // dump type  
    PMINIDUMP_EXCEPTION_INFORMATION ExceptionParam, // exception information  
    PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam, // user stream information  
    PMINIDUMP_CALLBACK_INFORMATION CallbackParam // callback information  
)
```

- Writes dump including the full stack and registers
- Microsoft provides symbol servers for system libraries
- Let you symbolicate the dump and analyze it
- A lot of information in ~ 100kb
- **Nothing similar existed on macOS**

- There is Google Breakpad/Crashpad of course:
 - Writes Windows minidumps on all systems
 - Need custom tools to analyze for Posix crashes (instead of lldb)
 - Now planning to let lldb support windows pdb format
- A lot of code (that you need to support if you include it!)
- not very powerful solution

- There is Google Breakpad/Crashpad of course:
 - Writes Windows minidumps on all systems
 - Need custom tools to analyze for Posix crashes (instead of lldb)
 - Now planning to let lldb support windows pdb format
- A lot of code (that you need to support if you include it!)
- not very powerful solution
- Mach-o file format is well documented, google "Mach-O File Format Reference"
- This includes the core file format
 - there are docs for ELF as well, but ELF core file is not standardized
(Check what gdb does)
- "all" we have to do is write mach-o core file only with stack memory

Out-of-process crash handling: Send task access rights

```
mach_port_t portBootstrap;
task_get_bootstrap_port(mach_task_self(), &portBootstrap);

// Lookup port opened by handler process
mach_port_t portChild;
bootstrap_look_up(portBootstrap, "port name", &portChild);

STcDumpMsg msg = {
    { MACH_MSGH_BITS_REMOTE(MACH_MSG_TYPE_COPY_SEND)
        | MACH_MSGH_BITS_COMPLEX, sizeof(msg), portChild },
    { 1 },
    // Message copying access rights to mach_task_self()
    { mach_task_self(), 0, 0, MACH_MSG_TYPE_COPY_SEND,
        MACH_MSG_PORT_DESCRIPTOR }
};
mach_msg(std::addressof(msg.header), MACH_SEND_MSG, sizeof(msg), 0,
    MACH_PORT_NULL, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

Out-of-process crash handling: Send task access rights

```
mach_port_t portBootstrap;
task_get_bootstrap_port(mach_task_self(), &portBootstrap);

// Lookup port opened by handler process
mach_port_t portChild;
bootstrap_look_up(portBootstrap, "port name", &portChild);

STcDumpMsg msg = {
    { MACH_MSGH_BITS_REMOTE(MACH_MSG_TYPE_COPY_SEND)
        | MACH_MSGH_BITS_COMPLEX, sizeof(msg), portChild },
    { 1 },
    // Message copying access rights to mach_task_self()
    { mach_task_self(), 0, 0, MACH_MSG_TYPE_COPY_SEND,
        MACH_MSG_PORT_DESCRIPTOR }
};
mach_msg(std::addressof(msg.header), MACH_SEND_MSG, sizeof(msg), 0,
    MACH_PORT_NULL, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

Out-of-process crash handling: Send task access rights

```
mach_port_t portBootstrap;
task_get_bootstrap_port(mach_task_self(), &portBootstrap);

// Lookup port opened by handler process
mach_port_t portChild;
bootstrap_look_up(portBootstrap, "port name", &portChild);

STcDumpMsg msg = {
    { MACH_MSGH_BITS_REMOTE(MACH_MSG_TYPE_COPY_SEND)
        | MACH_MSGH_BITS_COMPLEX, sizeof(msg), portChild },
    { 1 },
    // Message copying access rights to mach_task_self()
    { mach_task_self(), 0, 0, MACH_MSG_TYPE_COPY_SEND,
        MACH_MSG_PORT_DESCRIPTOR }
};
mach_msg(std::addressof(msg.header), MACH_SEND_MSG, sizeof(msg), 0,
    MACH_PORT_NULL, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
```

Out-of-process crash handling: Read thread state

```
mach_msg_type_number_t cThreads;
thread_array_t athread;

task_threads(task, &athread, &cThreads);

struct SThreadCommand {
    thread_command m_header;
    x86_thread_state m_threadstate;
    x86_float_state m_floatstate;
    x86_exception_state m_exceptionstate;
};

std::vector<SThreadCommand> vecthreadcmd;
// call thread_get_state for each thread/state pair
```

Out-of-process crash handling: Read thread state

```
mach_msg_type_number_t cThreads;
thread_array_t athread;

task_threads(task, &athread, &cThreads);

struct SThreadCommand {
    thread_command m_header;
    x86_thread_state m_threadstate;
    x86_float_state m_floatstate;
    x86_exception_state m_exceptionstate;
};

std::vector<SThreadCommand> vecthreadcmd;
// call thread_get_state for each thread/state pair
```

Out-of-process crash handling: Read stack memory

```
mach_vm_address_t pvBegin = 0;
mach_vm_size_t cb = 0;
vm_region_submap_info_64 vmregioninfo;
natural_t nDepth = 0;
mach_msg_type_number_t cbVMRegionInfo =
    VM_REGION_SUBMAP_INFO_COUNT_64;

std::vector<segment_command_64> vecsegment;
while(KERN_SUCCESS == mach_vm_region_recurse(
    task, &pvBegin, &cb, &nDepth, &vmregioninfo, &cbVMRegionInfo
)) {
    if(VM_MEMORY_STACK==vmregioninfo.user_tag) { // stack memory
        vecsegment.emplace_back(
            segment_command_64{LC_SEGMENT_64, ...}
        );
    }
    pvBegin += cb;
}
```

Out-of-process crash handling: Read stack memory

```
mach_vm_address_t pvBegin = 0;
mach_vm_size_t cb = 0;
vm_region_submap_info_64 vmregioninfo;
natural_t nDepth = 0;
mach_msg_type_number_t cbVMRegionInfo =
    VM_REGION_SUBMAP_INFO_COUNT_64;

std::vector<segment_command_64> vecsegment;
while(KERN_SUCCESS == mach_vm_region_recurse(
    task, &pvBegin, &cb, &nDepth, &vmregioninfo, &cbVMRegionInfo
)) {
    if(VM_MEMORY_STACK==vmregioninfo.user_tag) { // stack memory
        vecsegment.emplace_back(
            segment_command_64{LC_SEGMENT_64, ...}
        );
    }
    pvBegin += cb;
}
```

Out-of-process crash handling: Read stack memory

```
mach_vm_address_t pvBegin = 0;
mach_vm_size_t cb = 0;
vm_region_submap_info_64 vmregioninfo;
natural_t nDepth = 0;
mach_msg_type_number_t cbVMRegionInfo =
    VM_REGION_SUBMAP_INFO_COUNT_64;

std::vector<segment_command_64> vecsegment;
while(KERN_SUCCESS == mach_vm_region_recurse(
    task, &pvBegin, &cb, &nDepth, &vmregioninfo, &cbVMRegionInfo
)) {
    if(VM_MEMORY_STACK==vmregioninfo.user_tag) { // stack memory
        vecsegment.emplace_back(
            segment_command_64{LC_SEGMENT_64, ...}
        );
    }
    pvBegin += cb;
}
```

Out-of-process crash handling: Write to File

```
struct mach_header_64
+ std::vector<SThreadCommand>
+ std::vector<segment_command_64>
+ actual memory segments
```

- Resulting file can be loaded in lldb as core file
- As zip file ~100-200 Kb
- Need some additional meta data

```
<m_nThread val="0"/>
<m_vecmodule length="538">
  <elem>
    <m_pvStartAddress val="140735072636928"/>
    <m_strPath>/usr/lib/libz.1.dylib</m_strPath>
    <m_modver val="66059"/>
    <m_uuid val="db120508-3bed-37a8-b439-5235eab4618a"/>
  </elem>
```

Out-of-process crash handling: Backend

- Needs debug symbols for your builds
- Cached system binaries for macOS
- Symbol lookup: <https://lldb.llvm.org/use/symbols.html>
- Backend wraps lldb <https://lldb.llvm.org/design/sbapi.html>
 - load core file
 - lookup binaries/symbols and add them as modules
- Check out <https://github.com/think-cell/minidump>

1. Levels of Abstraction: Handling Files
2. Kernel Object Lifetimes: Interprocess Shared Memory
3. Common Tooling I: Text Internationalization
4. Common Tooling II: Error Reporting
5. **Moving to WebAssembly**

- think-cell ships with Google Chrome extension and web app
- What language to use?

- think-cell ships with Google Chrome extension and web app
- What language to use?
 1. No JavaScript
 2. TypeScript looked much better
 - somewhat type-safe, type definition libraries
 - **but** sharing code with C++ was impossible
 3. Emscripten looked interesting
 - Interfacing with JavaScript loses type-safety again
 - Lacking wrappers for standard APIs

```
auto xhr = emscripten::val::global("XMLHttpRequest").new_();  
xhr["open"]("GET", "http://google.com");
```

- think-cell ships with Google Chrome extension and web app
- What language to use?
 1. No JavaScript
 2. TypeScript looked much better
 - somewhat type-safe, type definition libraries
 - **but** sharing code with C++ was impossible
 3. Emscripten looked interesting
 - Interfacing with JavaScript loses type-safety again
 - Lacking wrappers for standard APIs

```
auto xhr = emscripten::val::global("XMLHttpRequest").new_();  
xhr["open"]("GET", "http://google.com");
```

... so we build our own "compiler"

Type definition libraries

```
interface Document extends Node, NonElementParentNode, DocumentOrShadowRoot {  
    readonly URL: string;  
  
    readonly activeElement: Element | null;  
  
    readonly anchors: HTMLCollectionOf<HTMLAnchorElement>;  
  
    title: string;  
  
    createElement<K extends keyof HTMLElementTagNameMap>(  
        tagName: K, options?: ElementCreationOptions  
    ): HTMLElementTagNameMap[K];  
}
```

<https://github.com/DefinitelyTyped/DefinitelyTyped>

Available for over 7000 JavaScript libraries, including standard DOM APIs

Powerful TypeScript compiler API:

```
function transform(file: string) : void {
    let program = ts.createProgram([file]);
    const sourceFile = program.getSourceFile(file);

    ts.forEachChild(sourceFile, node => {
        if (ts.isFunctionDeclaration(node)) {
            // do something
        } else if (ts.isVariableStatement(node)) {
            // do something else
        }
    });
}
```

typescripten — <https://github.com/think-cell/tcjs>

- Compiles typescript interface declarations to C++ interfaces
- i.e. type-safe, idiomatic calls to JavaScript libraries via emscripten

JavaScript:

```
document.title = "Hello World from C++";
```

C++:

```
using namespace tc;
js::document()->title(js::string("Hello World from C++!"));
```

```
namespace js {
    struct JsObject {
        emscripten::val m_emval;
    };

    struct Document : virtual Node, ... {
        auto URL() noexcept;
        auto activeElement() noexcept;
        auto title() noexcept;
        void title(string v) noexcept;
        // ...
    };
}

inline auto Document::title() noexcept { return m_emval["title"].template as<string>(); }
inline void Document::title(string v) noexcept { m_emval.set("title", v); }

inline auto document() noexcept {
    return emscripten::val::global("document").template as<Document>();
}
}
```

typescript is almost self-hosting, i.e., can compile typescript compiler interface definition

```
void transform(js::string const& file) {
    js::Array<js::string> arr(jst::create_js_object);
    arr->push(file);

    auto const program = js::ts::createProgram(arr, ...);
    auto const sourceFile = program->getSourceFile(file);

    js::ts::forEachChild(sourceFile,
        js::lambda(
            [](js::ts::Node jnodeChild) noexcept -> js::unknown {
                if(js::ts::isFunctionDeclaration(jnodeChild)) {
                }
            }
        )
    );
}
```

Declaration order does not matter in typescript

```
type FooBar = test.Foo | test.Bar;

declare namespace test {
    export interface Foo {
        a: string;
    }

    export interface Bar {
        b: number;
    }
}
```

String enums with custom marshaling

```
export enum FunnyEnum {  
    foo = "foo",  
    bar = "bar"  
}
```

String enums with custom marshaling

```
export enum FunnyEnum {
    foo = "foo",
    bar = "bar"
}
```

```
enum class FunnyEnum { foo, bar };

template<> struct MarshalEnum<FunnyEnum> {
    static inline auto const& Values() {
        static tc::unordered_map<FunnyEnum, unknown> vals{
            {FunnyEnum::foo, string("foo")},
            {FunnyEnum::bar, string("bar")}
        };
        return vals;
    }
};
```

Function-callbacks as ref-counted objects

This is what we want:

```
document.onclick(OnClickHandler)
```

This is what we have:

```
struct S {
    JS_MEMBER_FUNCTION(S, OnClick, void, (js::MouseEvent ev)) {
        js::document()->title(js::string("Hello World from C++!"));
    }

    void AddHandler() {
        js::document()->onclick(OnClick);
    }
};
```

Function-callbacks as ref-counted objects

```
JsFunction<void (js::MouseEvent)> OnClick{&OnClickWrapper, this};  
// 1. Create RAII wrapper OnClick  
// 2. Calls into JavaScript, creates JS function object,  
// stores C++ function ptr and this ptr in JS object  
// 3. Store JavaScript function object as emscripten::val  
  
emscripten::val Call(PointerNumber iFunctionPtr, PointerNumber iArg0,  
    emscripten::val emvalThis, emscripten::val emvalArgs) noexcept { ... }  
// 4. JavaScript function object passes function pointer back to generic C++ function  
// 5. Casts function pointer to correct signature and calls it  
  
static emscripten::val OnClickWrapper(void* pvThis, emscripten::val const& emvalThis,  
    emscripten::val const& emvalArgs) noexcept  
{  
    // 6. Cast this pointer, unpack arguments from emvalArgs and call OnClickImpl  
}  
  
void OnClickImpl(js::MouseEvent ev) noexcept /* ... user code */
```

Open Challenges:

- Generics

```
interface HTMLCollectionOf<T extends Element> extends HTMLCollectionBase {  
    item(index: number): T | null;  
}
```

- Indexed access types

```
interface DocumentEventMap {  
    "click": MouseEvent;  
    "keydown": KeyboardEvent;  
}  
addEventListener<K extends keyof DocumentEventMap>(  
    type: K, listener: (this: Document, ev: DocumentEventMap[K]) => any, ...  
): void;
```

- Better performance for calls e.g. by generating static linked JS functions
 - string lookups for properties and functions are slow

Thank you!

Now to your questions!

Sebastian Theophil, think-cell Software, Berlin

stheophil@think-cell.com