Windows, macOS and the Web

Lessons from cross-platform development at think-cell

Sebastian Theophil, think-cell Software, Berlin
stheophil@think-cell.com

The Problem

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

Agenda

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

Levels of Abstraction

Levels of Abstraction

- How many ways to rename a file?

Levels of Abstraction

- How many ways to rename a file?

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

int renamex_np(char const* from, char const* to, unsigned int flags)
int copyfile(char const* from, char const* to, copyfile_state_t state,
    The copyfile_flags_t flags)

- [NSFileManager replaceItemAtURL:withItemAtURL:backupItemName:
    options:resultingItemURL:error:]

int rename(char const* old, char const* new)

bool QFile::rename(QString const& newName)

void boost::filesystem::rename(const path& old_p, const path& new_p)

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

Levels of Abstraction

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

Levels of Abstraction

- 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

Levels of Abstraction

- 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
  - create a user application settings file
  - create a temporary file that is automatically deleted but which can be opened by another application
  - download a file to a cache in thread-safe way
  - 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
```

Levels of Abstraction

- Cross-platform interfaces need to have well-defined, strong semantics
- Weak semantics lead to subtle errors
  - Warning sign: Having to look at the implementation
- Too high-level: You miss a chance to unify code
- 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`)

Agenda

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

Kernel Object Lifetimes

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

Kernel Object Lifetimes

- 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`

Kernel Object Lifetimes

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);
```

Kernel Object Lifetimes

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);
```

Kernel Object Lifetimes

- **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**

Kernel Object Lifetimes

- **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

Kernel Object Lifetimes

- This seems to cause problems occasionally:

stackoverflow Products Search…

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS What's this?

Free 30 Day Trial

Remove posix shared memory when not in use?

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

15

3

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.

Kernel Object Lifetimes

- This seems to cause problems occasionally:

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

6

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?

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

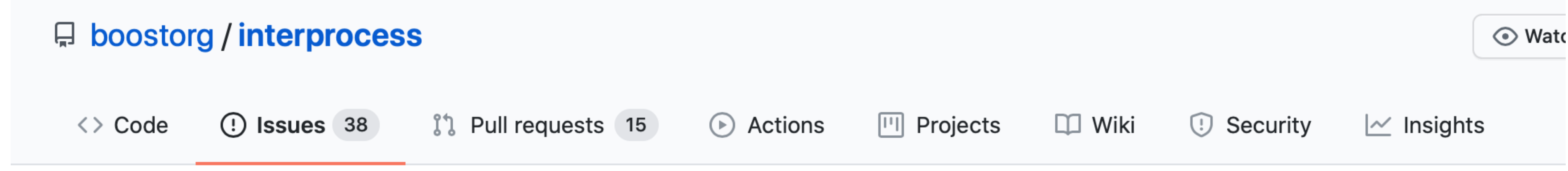
asked Nov 9 '09 at 10:47
Vivek
423 ●1 ●3 ●10

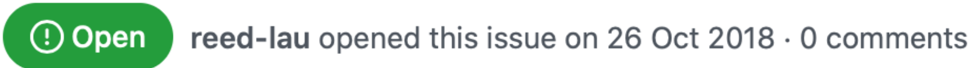
add a comment

start a bounty

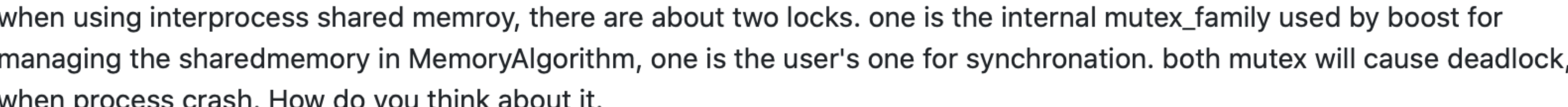
Kernel Object Lifetimes

- This seems to cause problems occasionally:





reed-lau commented on 26 Oct 2018



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.

Kernel Object Lifetimes

- This seems to cause problems occasionally:

stackoverflow   Products   Search…   4,613 • 20

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS   What's this?

Free 30 Day Trial

how to free managed_shared_memory after a program crash: what are effective techniques to use during debugging?

Asked 4 years, 4 months ago   Active 4 years, 3 months ago   Viewed 460 times

1

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.

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.

So, after such a program crash, when I attempt to run my application again, when my code tires to allocate a new segment of shared memory I see messages like:

The Overflow B

The Overfl

Featured on M

Respondir
commitme

What shou
[*vcf*] tag

Hot Meta Posts

36 Rename th

Kernel Object Lifetimes

- 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.

Kernel Object Lifetimes

- 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**

Kernel Object Lifetimes

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

Kernel Object Lifetimes

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

Kernel Object Lifetimes

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

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_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, O_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

Kernel Object Lifetimes

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

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_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, O_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

Kernel Object Lifetimes

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

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_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, O_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

Kernel Object Lifetimes

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

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_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, O_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

Kernel Object Lifetimes

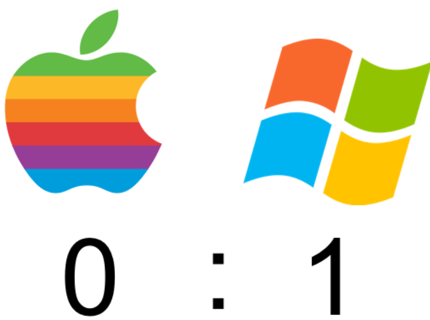
- Introduces `basic_managed_nonpersistent_shared_memory`
- `nonpersistent_shared_memory_object::priv_open_or_create`:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_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, O_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

Kernel Object Lifetimes

- 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 is left to the user when cross-platform toolkits fail



Agenda

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

Mouse Events

- Superficially, Windows and macOS have a similar event handling architecture
- Windows sends messages to windows:

```
struct CMyWindow: ATL::CWindowImpl<CMyWindow>
{
    // window message map
    BEGIN_MSG_MAP(CMyWindow)
        MESSAGE_HANDLER(WM_MOUSEMOVE, OnMouseMove)
        MESSAGE_HANDLER(WM_LBUTTONDOWN, OnButtonDown)
        MESSAGE_HANDLER(WM_LBUTTONDBLCLK, OnDoubleClick)
    END_MSG_MAP()
    ...
    LRESULT OnMouseMove(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
    LRESULT OnButtonDown(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
    LRESULT OnDoubleClick(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
};
```

Mouse Events

- Superficially, Windows and macOS have a similar event handling architecture
- macOS handles the messages and calls event handlers directly:

```
@interface MyView : NSView {
    - (void)mouseMoved:(NSEvent*)nsevent;
    - (void)mouseDragged:(NSEvent*)nsevent;
    - (void)mouseDown:(NSEvent*)nsevent;
}
```

Mouse Events

- Superficially, Windows and macOS have a similar event handling architecture
- macOS handles the messages and calls event handlers directly:

```
@interface MyView : NSView {
    - (void)mouseMoved:(NSEvent*)nsevent;
    - (void)mouseDragged:(NSEvent*)nsevent;
    - (void)mouseDown:(NSEvent*)nsevent;
}
```

- The semantical differences are large though

Mouse Events

Windows	macOS
Coordinates relative to client-area of the window/component	Coordinates relative to top-level window
Single or double click	`-[NSEvent clickCount]`
`WM_MOUSELEAVE`, `WM_MOUSEENTERED` are opt-in events.	`-[NSResponder mouseEntered]` `-[NSResponder mouseExited]`
No distinction between `WM_MOUSEMOVE` and drag events.	`-[NSResponder mouseMoved]` `-[NSResponder mouseDragged]`

Mouse Events

Windows	macOS
`SetCapture/ReleaseCapture` to receive mouse messages after mouse has exited the window. `WM_CAPTURECHANGED` `WM_CANCELMODE`	Capture is automatic
Mouse message order can be "surprising", see `QWindowsMouseHandler::translateMouseEvent`	`-[NSResponder mouseMoved]` `-[NSResponder mouseDown]` `-[NSResponder mouseDragged]` `-[NSResponder mouseUp]`

Mouse Events

Windows	macOS
`SetCapture/ReleaseCapture` to receive mouse messages after mouse has exited the window. `WM_CAPTURECHANGED` `WM_CANCELMODE`	Capture is automatic
Mouse message order can be "surprising", see `QWindowsMouseHandler::translateMouseEvent`	`-[NSResponder mouseMoved]` `-[NSResponder mouseDown]` `-[NSResponder mouseDragged]` `-[NSResponder mouseUp]`

The macOS model is much saner, offers strong guarantees.

Strong guarantees are good!

Mouse Events

(Simplified) Mouse Message State Machine

1. `WM_MOUSEMOVE`
   - keep track of mouse window
   - `TrackMouseEvent` registers for `WM_MOUSELEAVE`

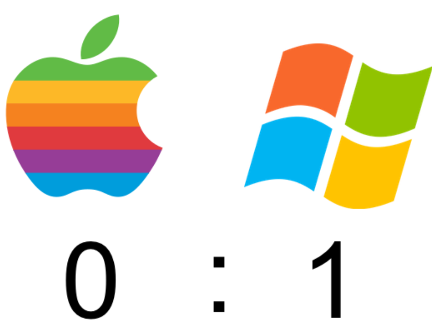
2. `WM_XBUTTONDOWN`
   - keep track of pressed button
   - ignore all other button presses
   - call `SetCapture` to receive messages

3. `WM_XBUTTONUP`
   - *Pressed* button is released, ignore others
   - call `ReleaseCapture`

3. `WM_CANCELMODE` / `WM_CAPTURECHANGED`
   - Clear mouse state

Mouse Events

- Aiming for strong and identical semantics
  ... strong semantics **implies strong invariants**
  ... must hold on each operating system
  ... unify the number of states your app may have



Agenda

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

Text Internationalization

- 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 TRANSLOOKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSLOOKUP(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 TRANSLOOKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSLOOKUP(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 TRANSLOOKUP(String, Context) ...
#define TRANSLATECTX(String, Context) \
    translate(TRANSLOOKUP(String, Context))

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

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;
};

#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)
```

Text Internationalization

- 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

Text Internationalization

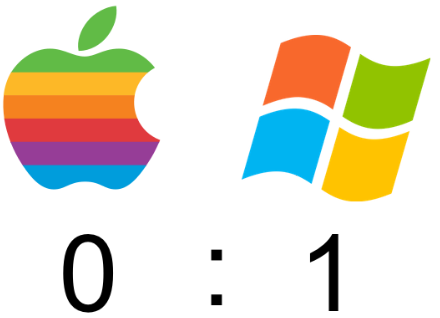
```
namespace StaticMurmurHash {
    using uint128_t = std::pair<std::uint64_t, std::uint64_t>;

    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

- Aiming for strong and identical semantics

  ... that includes external tools in your build process!



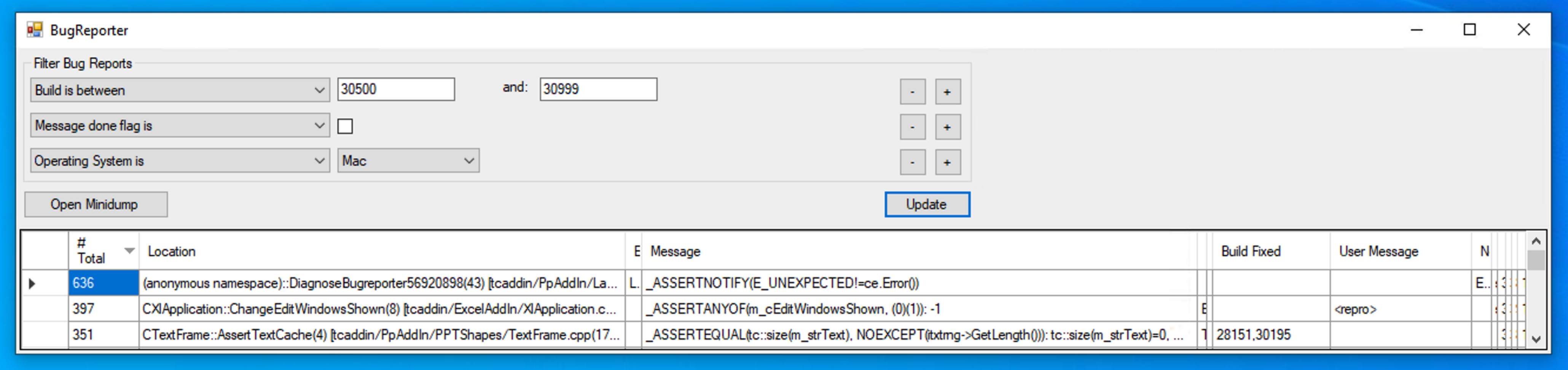
Agenda

1. Levels of Abstraction: Handling Files
2. Kernel Object Lifetimes: Interprocess Shared Memory
3. Diverging OS Behavior: Handling Mouse Events
4. Common Tooling I: Text Internationalization
5. **Common Tooling II: Error Reporting**
6. 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**

Error Reporting

The core of this functionality on Windows:

```
BOOL MiniDumpWriteDump(
  HANDLE                            hProcess,
  DWORD                             ProcessId,
  HANDLE                            hFile,
  MINIDUMP_TYPE                     DumpType,
  PMINIDUMP_EXCEPTION_INFORMATION   ExceptionParam,
  PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,
  PMINIDUMP_CALLBACK_INFORMATION    CallbackParam
)
```

- 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**

Error Reporting

- 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

Error Reporting

- 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

Error Reporting

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);
```

Error Reporting

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);
```

Error Reporting

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);
```

Error Reporting

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
```

Error Reporting

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
```

Error Reporting

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;
}
```

Error Reporting

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;
}
```

Error Reporting

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;
}
```

Error Reporting

Out-of-process crash handling: Write to File

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

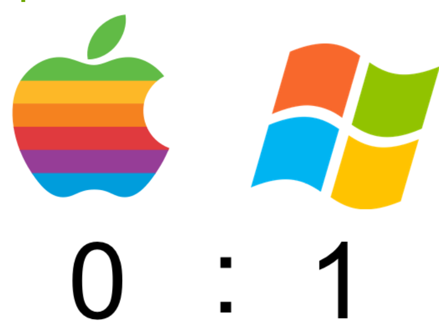
- Resulting file can be loaded in lldb as core file
- As zip file ~100-200 Kb
- Need some additional meta data

```
<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>
```

Error Reporting

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



Agenda

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

WebAssembly

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

WebAssembly

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

1. JavaScript was a hard no
2. TypeScript looked much better
    - somewhat type-safe, type definition libraries https://github.com/DefinitelyTyped/DefinitelyTyped
    - **but** sharing code with C++ was impossible
3. Emscripten looked interesting
    - Interfacing with JavaScript loses type-safety again

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

WebAssembly

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

1. JavaScript was a hard no
2. TypeScript looked much better
   - somewhat type-safe, type definition libraries https://github.com/DefinitelyTyped/DefinitelyTyped
   - **but** sharing code with C++ was impossible
3. Emscripten looked interesting
   - Interfacing with JavaScript loses type-safety again

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

... so we build our own compiler

Type definition libraries

```
interface HTMLElement extends Element, GlobalEventHandlers, ... {
    hidden: boolean;
    innerText: string;
    readonly offsetParent: Element | null;

    click(): void;

    ...
}
```

WebAssembly

+ 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
        }
    });
}
```

WebAssembly

tcjs — https://github.com/think-cell/tcjs

- Compiles typescript interface declarations to C++ interfaces
- **i.e. type-safe calls to JavaScript libraries via emscripten**
- Almost self-hosting, i.e., compiling typescript compiler interface
- Still missing typescript language features, but already usable
- Originally master thesis of Egor Suvorov at think-cell
- Check it out!

WebAssembly

Type-safe calls to JavaScript/TypeScript libraries via emscripten

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

Check out https://github.com/think-cell/tcjs

Lots of interesting work to do!

Conclusion

No premature unification in code

Unify object lifetimes across Operating Systems

Maintain cross-platform invariants with state machines

Make your build tools and backends cross-platform

Conclusion

No premature unification in code

Unify object lifetimes across Operating Systems

Maintain cross-platform invariants with state machines

Make your build tools and backends cross-platform

Always use C++ 🤟

Thank you!

Now to your questions!

Sebastian Theophil, think-cell Software, Berlin
stheophil@think-cell.com