COMPILE-TIME FUNCTION CALL INTERCEPTION TO MOCK FUNCTIONS IN C/C++

Gábor Márton, Zoltán Porkoláb

Ericsson Hungary Ltd., Eötvös Loránd University, Budapest

martongabesz@gmail.com, zoltan.porkolab@ericsson.com,

EuroLLVM 2018

AGENDA

- Problem Definition
- Existing Solutions
- Our Solution
- Future Work

PROBLEM DEFINITION

```
class FooServer {
  std::mutex m;

public:
    int process(int) {
        if (m.try_lock()) {
            // ...
        } else {
            // ...
        }
    }
};
```

PROBLEM DEFINITION

```
struct IMutex {
  virtual void lock() = 0;
  virtual void unlock() = 0;
  virtual bool try_lock() = 0;
};
struct RealMutex : IMutex { /*...*/ };
struct StubMutex : IMutex { /*...*/ };
```

PROBLEM DEFINITION

```
class FooServer {
  IMutex& m;
public:
  FooServer(IMutex &m) : m(m) {}
  int process(int) { /*...*/ }
};
```

```
int main() {
  RealMutex m;
  FooServer s(m);
  // Real usage of s
  // ...
}
```

```
void test() {
  StubMutex m;
  FooServer s(m);
  ASSERT_EQUALS(s.process(1), -1);
}
```

PROBLEM DEFINITION

```cpp
template <typename Mutex>
class FooServer {
  Mutex m;
public:
  int process(int) { /*...*/ }
};
```

```cpp
int main() {
  FooServer<RealMutex> s;
  // Real usage of s
  // ...
}
```

```cpp
void test() {
  FooServer<StubMutex> s;
  // Test code from here ...
}
```

NON-INTRUSIVE TESTS

- Transparent to the source code of the software under test
- No source code change in the production code

- Useful for
  - Keep the original structure
  - Test legacy software
  - White box testing

TOOLS FOR NON-INTRUSIVE TESTS

- LD_PRELOAD
  - Load an other library for testing
  - Inlining?
  - Static libs?
- Binary instrumentation (Intel PIN)
  - Inlining?
  - Mangled names?

- `finstrument_functions`
  - Instruments the body of each function
    - Emitted hook functions
      - `__cyg_profile_func_enter`
      - `__cyg_profile_func_exit`
  - Replace functions?

MOTIVATING EXAMPLE

```
#include "FooServer.hpp"

bool try_lock_result;
bool fake_mutex_try_lock(std::mutex *self) {
  return try_lock_result;
}

void test() {
  SUBSTITUTE(std::mutex::try_lock, fake_mutex_try_lock);
  FooServer s;
  try_lock_result = false;
  ASSERT_EQUALS(s.process(1), -1);
}
```

OUR SOLUTION

- Compile-time instrumentation
  - Enabled when compiling test code
  - Similar to `finstrument-functions`, but
- *Call expression* instrumentation!
  - Emits a hook function
    - Decide the target function
    - Call the target function
  - The body of the called function is left intact
  - No need to recompile system libs or 3rd party shared libs
  - Inlining disabled

ARCHITECTURE

- Runtime Library
  - C API
  - C++ API
  - Lookup
- Frontend
  - Language specific
- Backend
  - Modifies LLVM IR
    - CodeGen
    - LLVM pass (future)

BACKEND

› Call expression instrumentation!

› Original CallExpr:

```
foo(p1, p2);
```

› Instrumented CallExpr:

```
char* funptr = __fake_hook(&foo);
if (funptr)
    funptr(p1, p2);
else
    foo(p1, p2);
```

RUNTIME LIBRARY

- Lookup
  - Hash Map

Original	Replacement
&foo	&fake_foo
&bar	&fake_bar
…	…

  - Shadow Memory (like sanitizers)

```
[0x7f0000000000, 0x7fffffffffff] || HighMem
[0x120000000000, 0x19ffffffffff] || HighShadow
[0x020000000000, 0x11ffffffffff] || LowShadow
[0x000000000000, 0x01ffffffffff] || LowMem
```

RUNTIME LIBRARY

› C API

```
void foo();
void fake_foo();
_substitute_function((const char*)&foo, (const char*)&fake_foo);
SUBSTITUTE(foo, fake_foo);
```

› C++ API

```
struct X {  virtual void foo();   int bar(int); int bar(char);  };
void X_fake_foo(X* self);
SUBSTITUTE(X::foo, X_fake_foo);
int X_fake_bar_i(X* self);
SUBSTITUTE(int(int), X::bar, X_fake_bar_i);
```

C++ FRONTEND

- New Unary Expression
  - for Virtual Functions
  - Overload resolution

```
void foo();
auto p = & foo; // void (*)()
auto q = __function_id foo; // void (*)()

struct X { void foo(); virtual void bar(); };
auto p0 = & X::foo;              // void (X::*)()
auto p1 = __function_id X::foo; // void (*)()
auto p2 = & X::bar;              // void (X::*)()
auto p3 = __function_id X::bar; // void (*)()
```

FUTURE WORK

- Make it faster
  - Do not instrument all functions
  - XRay like noops

- Alternative approach: replace on the AST level
  - Reuse ASTImporter

- Gábor Márton
- Call Expression instrumentation to mock C/C++ functions
- Working prototype
  - https://github.com/martong/finstrument_mock

ERICSSON